THE UNIVERSITY
of EDINBURGH

Unsupervised Learning with Neural Latent Variable Models

Charlie Nash

Doctor of Philosophy

Institute for Adaptive and Neural Computation
School of Informatics
University of Edinburgh

2020

Abstract

Latent variable models assume the existence of unobserved factors that are responsible for generating observed data. Deep latent variable models that make use of neural components are effective at modelling and learning representations of data. In this thesis we present specialised deep latent variable models for a range of complex data domains, address challenges associated with the presence of missing-data, and develop tools for the analysis of the representations learned by neural networks.

First we present the shape variational autoencoder (ShapeVAE), a deep latent variable model of part-structured 3D objects. Given an input collection of part-segmented objects with dense point correspondences the ShapeVAE is capable of synthesizing novel, realistic shapes, and by performing conditional inference can impute missing parts or surface normals. In addition, by generating both points and surface normals, our model enables us to use powerful surface-reconstruction methods for mesh synthesis. We provide a quantitative evaluation of the ShapeVAE on shape-completion and test-set log-likelihood tasks and demonstrate that the model performs favourably against strong baselines. We demonstrate qualitatively that the ShapeVAE produces plausible shape samples, and that it captures a semantically meaningful shape-embedding. In addition we show that the ShapeVAE facilitates mesh reconstruction by sampling consistent surface normals.

Latent variable models can be used to probabilistically fill-in missing data entries. The variational autoencoder architecture (Kingma and Welling, 2014; Rezende et al., 2014) includes a recognition or encoder network that infers the latent variables given the data variables. However, it is not clear how to handle missing data variables in these networks. The factor analysis (FA) model is a basic autoencoder, using linear encoder and decoder networks. We show how to calculate exactly the latent posterior distribution for the FA model in the presence of missing data, and note that this solution exhibits a non-trivial dependence on the pattern of missingness. We also discuss various approximations to the exact solution. Experiments compare the effectiveness of various approaches to imputing the missing data.

Next, we present an approach for learning latent, object-based representations from image data, called the multi-entity variational autoencoder (MVAE), whose prior

and posterior distributions are defined over a set of random vectors. Object-based representations are closely linked with human intelligence, yet relatively little work has explored how object-based representations can arise through unsupervised learning. We demonstrate that the model can learn interpretable representations of visual scenes that disentangle objects and their properties.

Finally we present a method for the analysis of neural network representations that trains autoregressive decoders called inversion models to express a distribution over input features conditioned on intermediate model representations. Insights into the invariances learned by supervised models can be gained by viewing samples from these inversion models. In addition, we can use these inversion models to estimate the mutual information between a model's inputs and its intermediate representations, thus quantifying the amount of information preserved by the network at different stages. Using this method we examine the types of information preserved at different layers of convolutional neural networks, and explore the invariances induced by different architectural choices. Finally we show that the mutual information between inputs and network layers initially increases and then decreases over the course of training, supporting recent work by Shwartz-Ziv and Tishby (2017) on the information bottleneck theory of deep learning.

Lay Summary

This thesis focuses on models that explain observed data in terms of unobserved factors. These models are commonly known as latent variable models. For instance, in a healthcare setting, a patient's elevated blood pressure may be explained by the incidence of an unobserved medical condition, such as heart disease. We focus on models that use complex transformations known as neural networks to map hidden states to data values, and that can be used to generate new data examples. In particular we design specialised models for different domains, address challenges associated with corrupted or incomplete datasets, and use these models as an analysis tool for other neural networks.

In Chapter 3 we present a latent variable model of 3D objects, where the goal is to take a collection of examples from a particular object class – e.g chairs or planes – and to build a model that learns the characteristic features of that class. In doing so we can generate new instances, which enables the automatic creation of 3D objects for digital environments.

Models with unobserved factors typically rely on methods that infer what hidden factors are associated with a given observation. This inference can either be exact or approximate, with approximate solutions usually being less computationally expensive. In Chapter 4 we provide an analysis of the limitations of certain approximation methods, and characterise their performance empirically.

In recent years a significant thread of machine learning research has focused on automatically representing objects along distinct directions of variability. For instance in order to represent a chair in a visual scene we might decompose it into a number of factors including rotation, colour and scale. The challenge is to do this in an unsupervised way: without any access to true labels for the object attributes. The majority of research has focused on visual scenes with a single object, but standard methods break down when applied to multi-object scenes. In Chapter 5 we address the task of obtaining factored representations of objects in multi-object scenes. Our key contribution is an inference mechanism that attends to the spatial locations in an image that are most informative.

Deep neural networks consist of a hierarchy of layers, that each represent the input data in different ways. The type of information that a network extracts from its input is of significant interest, helping us to understand why deep learning works,

and potentially informing our design choices. In Chapter 6 we present a method for analysing neural network layers, that uses another network to"invert" the layers back to the input space. In doing so we can assess what types of information have been extracted by the network.

Acknowledgements

I would like to thank Chris my supervisor for his support and guidance. Chris has been a dedicated and wise advisor, and has instilled in me a research perspective that I take with me.

I would also like to thank the Centre of Doctoral Training in Data Science and the Edinburgh School of Informatics. Together, they provided a fantastic environment for my PhD research. I am grateful to have met many inspirational people there, and thank the management and administrative staff that contribute so much to the school.

Finally, I would like to thank my friends and family for their support and advice. In particular my parents Penny and Craig, and my brother Jeremy. Most of all my partner Jess, who has been a continuous source of support and inspiration during my PhD years.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Charlie Nash*)

Table of Contents

Chapter 1

Introduction

One of the most important challenges in machine learning is modelling high-dimensional data. The world is full of this data: Images, audio, text, video, weather measurements, health indicators and financial records. This data is high-dimensional in the sense that each observation is multi-faceted. For instance, a single document contains many characters, an image consists of many pixels. We want to be able to get a handle on this data; to understand its structure, to make predictions about it, or even to try and understand how humans process it.

One of the key tools for working with data is statistical modelling, where we build models that describe how data might be generated. If our statistical model is flexible enough to capture the relationships of interest, and if we do a good job of estimating its parameters, then we can use the model to make predictions. By analyzing the fitted parameters we can try and understand how the variables of interest interact with one another. For data with a small number of dimensions, it is relatively straightforward to specify these models: We can choose a joint distribution over the variables, using standard parametric families such as Gaussians, or categorical distributions. For example, we could model the joint distribution of people's height and weight using a two-dimensional Gaussian distribution, which would allow us to capture the positive linear relationships between the variables. Or if we want to do object classification in images, we can model this using a categorical distribution over the target classes, with parameters that are regressed from the input image using a neural network. In both these cases, the distribution of interest is over a small number of variables, and so

distributional choices are relatively simple. But specifying statistical models for high-dimensional data is a significant challenge. For data with hundreds of variables, simple parametric families are unable to capture all the complex dependencies, and we need to look to alternative options.

One of the most prominent classes of models for high-dimensional data is latent-variable models, and they are a central focus of this thesis. Latent variable models make the assumption that the observed data is generated by interactions with some unobserved variables. For example, in a healthcare setting, we might observe a drop in a patient's blood pressure, and see in their notes that they have recently had a fever. Does the fever directly affect the patient's blood pressure, or vice versa? Actually, both may be better explained by an unobserved factor, like sepsis. Here, sepsis is a latent variable that explains the observed health indicators. Typically, latent variable models are specified such that the dependence among observed variables is simplified, *if* we know the values of the latent variables. For example, the observed variables are often assumed to be conditionally independent given the latent variables. In the previous example we might assume that blood pressure and fever are independent, given that the patient has sepsis.

Latent variable models are a powerful tool for modelling data. Statistical dependencies among the data variables are induced by the shared latent variables that generate them. This enables us to model the complex relationships between high dimensional data via a transformation of latent variables. By incorporating neural network components into our latent variable models, for instance by assuming that the data is generated by transforming latent variables with a neural network, we enable a rich class of transformations, that can represent complex data distributions. In addition, latent variable models enable us to specify certain kinds of structure *a priori*, such as conditional independence between certain variables, or neural architectures. We can therefore explicitly incorporate what we do know about the data via a partial specification of a data-generating process, and let the expressive power of deep learning fill in the gaps. Training latent variable models has historically been challenging, because of the difficulty in inferring the unobserved latent variables, however, in recent years amortized variational inference techniques with deep recognition networks have come to the fore, and now training these models is relatively straightforward (Kingma and Welling, 2014; Rezende et al., 2014).

This thesis focuses on the use of latent variable models to model and analyze rich high-dimensional datasets. We design specialized versions of these models in complex domains, analyze the properties and limitations of existing inference methods, and use them as analysis tools in other learning settings. We don't claim that latent variable models are *better* than alternatives, just that they present mechanisms for us to insert structure into our models, and that this can be useful for representation learning, or for analysis, or as a way to improve data-efficiency through the introduction of an inductive bias. The following sections introduce the main research areas we focus on in this thesis.

3D object generation. Latent variable models can be used for a wide range of applications, including the automated creation of digital content, like images (Kingma and Welling, 2014; Gulrajani et al., 2017), videos (Denton and Fergus, 2018), or audio signals (van den Oord et al., 2017). By building models of this data, that can propose or complete partial inputs, we can enhance the content creation process. With the increasing popularity of gaming, as well as virtual and augmented reality, an increasing challenge is to generate diverse but plausible virtual worlds: objects, characters, people, landscapes, structures etc. While various procedural generation methods have been proposed, they tend to operate in limited settings, such as terrain generation, or road networks (Hendrikx et al., 2013). But generative models that learn about the variability present in natural and man-made scenes have the potential to surpass these limitations. In Chapter 3, we demonstrate one such model, the ShapeVAE, that learns to generate point representations of 3D objects directly from data. By equipping the ShapeVAE with latent variables, we are able to capture a complex distribution over a large number of data variables, while learning an interpretable latent representation.

Missing data. An issue that frequently occurs when working with real data is missing observations. For instance electronic health records may be incomplete due to missed appointments, or weather data may be missing because of sensor failure. In many cases we may want to estimate the values of the missing observations, in order to perform further analysis. The use of probabilistic models for data imputation is promising, and latent variable models enable us to scale this approach to high-dimensional data distributions. In Chapter 4 we analyze approaches to dealing with missing data for the factor analysis latent variable model, and compare empirically the effectiveness of these methods.

Disentangled object representations. A key concept in machine learning is data representation, and in this thesis we will think about latent variable models as a means to build and analyze data representations. Representations of the data are transformations that expose salient features, making them useful for downstream tasks. What makes a representation good is an open question, but a common answer is that a good representation should disentangle the factors of variation present in the data (Bengio et al., 2013). For instance, when we observe an image of a car, we understand that the scene can be decomposed into the colour and shape of the car, the 3D pose of the car, the lighting, the camera position, etc. Another way to think about this is in terms of invariances: a disentangled representation is one for which each feature is invariant to changes in the others (Higgins et al., 2018). The question is, can we learn to decompose these scenes in an unsupervised way? In fact, there are questions about whether this is theoretically possible in general, for the same reasons that identifiability of parameters is not always possible in factor analysis models (Anderson and Rubin, 1956). However, there have been some successful attempts using latent variable models, with various methods recovering meaningful factors of variation in visual scenes (Higgins et al., 2017; Chen et al., 2016). Much of this work is applied to scenes consisting of single objects, however it can be challenging to decompose scenes consisting of multiple objects, as existing methods will tend to encode information about distinct objects in the same latent variables. In Chapter 5, we make progress towards disentangling across objects, with the multi-entity variational autoencoder (MVAE), that uses a structured inference process to encode separate objects into separate latent variables.

Analysis of learned representations. While structured latent variable models like the MVAE are a promising method for representation learning, the most successful large-scale representation learning methods use supervised learning. A classic example is in computer vision, where subsequent layers of a deep CNN, trained to do object classification, contain information at different levels of abstraction, from edges and colour blobs at the lowest layers, to textures and object-parts, and eventually to object-like representations at the highest levels of abstraction (Zeiler and Fergus, 2014). These representations often transfer well to other image processing tasks, like semantic segmentation (Chen et al., 2018), or captioning (Vinyals et al., 2015). The nature of the representations learned

by deep neural networks is of significant interest. For instance, what invariances do they have? And how much information about the inputs is retained, and how much is discarded? Does this change during the training process? In Chapter 6 we propose a way of answering these questions: By interpreting the intermediate representations in a model as latent variables, and by training decoder models that can invert the hidden representations back to the input space. In doing so we are able to visualize and quantify the information contained in network representations.

1.1 List of contributions

In **Chapter 3** we introduce the **shape variational autoencoder** (ShapeVAE), a deep generative model of 3D object shape, that captures discrete part structures as well as continuous shape variability. The ShapeVAE employs a part-structured encoder-decoder architecture, that learns both local part representations as well as global structural representations. The ShapeVAE is capable of synthesizing plausible and novel 3D objects, and through conditional inference enables imputation of missing parts or surface normals. We provide a quantitative evaluation of the ShapeVAE on shape-completion and test-set log-likelihood tasks, and demonstrate that the model performs favourably against strong baselines. This work was published in the following paper:

- Nash, C. and Williams, C. K. I. (2017). The shape variational autoencoder: A deep generative model of part-segmented 3d objects. In *Computer Graphics Forum*, volume 36, pages 112. Wiley Online Library.

In **Chapter 4** we present an analysis of approaches to handling latent-variable inference in the presence of **missing data**, for the widely used factor analysis model. In particular we show that exact inference in the presence of missing data requires a distinct matrix inversion for each pattern of missingness. In addition we provide an empirical comparison of a range of approximate methods for inference in the presence of missing data, using data imputation as the benchmark task. This chapter is adapted from the article:

- Williams, C. K. I., Nash, C. and Nazábal, A. (2018). Autoencoders and probabilistic inference with missing data: An exact solution for the factor

analysis case. Available at arXiv:1801.03851.

The main ideas and theoretical analysis for this chapter were contributed by Chris Williams, with input from myself and Alfredo Nazábal. I additionally contributed empirical results, as well as an analysis of these results.

In **Chapter 5** we present an object-centric model of 2D scenes, the **multi-entity variational autoencoder** (MVAE). The MVAE makes use of a novel informational attention mechanism that enables us to encode objects in different latent vectors. We demonstrate that the model can learn interpretable representations of visual scenes that disentangle objects and their properties. The MVAE is effective at localising objects even in the presence of significant overlap, and can be extended to scenes with large numbers of objects. This work was presented at the 2017 NeurIPS workshop on 'Learning Disentangled Features' as:

- Nash, C., Eslami, S. A., Burgess, C., Higgins, I., Zoran, D., Weber, T., and Battaglia, P. (2017). The multi-entity variational autoencoder

In **Chapter 6** we present a method for feature interpretation that makes use of recent advances in autoregressive density estimation models to invert model representations. We show how generative **inversion models** can be trained to express a distribution over input features conditioned on intermediate model representations. We further show how inversion models can be used to estimate the mutual information between a model's inputs and its intermediate representations, thus quantifying the amount of information preserved by the network at different stages. Using this method we examine the types of information preserved at different layers of convolutional neural networks, and explore the invariances induced by different architectural choices. Finally we show that the mutual information between inputs and network layers initially increases and then decreases over the course of training, supporting recent work by Shwartz-Ziv and Tishby (2017) on the information bottleneck theory of deep learning. This work was published in the following paper:

- Nash, C., Kushman, N., and Williams, C. K. (2019). Inverting supervised representations with autoregressive neural density models. In *AISTATS*, Proceedings of Machine Learning Research

1.2 Thesis structure

In Chapter 2 we present background material for the subsequent chapters. In particular we focus on latent variable models and the variational autoencoder. Chapters 3, 4, 5, 6 are each adapted from the papers described above. Finally in Chapter 7 we conclude our work and discuss future research directions in the context of recent developments in the field.

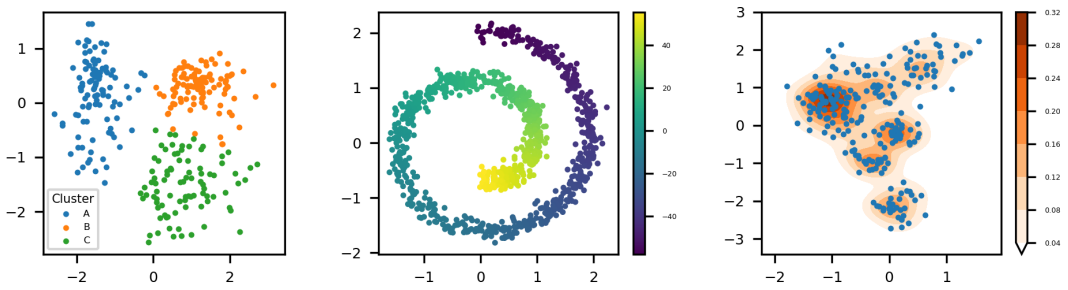
Chapter 2

Background

In this chapter we review approaches to unsupervised learning using latent variable models. We first introduce unsupervised learning (Section 2.1) before discussing latent-variable models (Sections 2.2, 2.3), and autoregressive models (Section 2.4). In particular we discuss classic linear-subspace models (Section 2.2.1), exact and approximate inference methods (Sections 2.2.2, 2.2.3), deep latent variable models (Section 2.3), the variational autoencoder (Section 2.3.2) and autoregressive generative models (Section 2.4).

2.1 Unsupervised learning

The task of unsupervised learning is to uncover structure in data, without using human-provided supervision as a guide to what is salient or interesting about particular observations. When doing unsupervised learning we seek to explain or analyze our data, or to provide useful inputs for further applications. In practice there exists a varied body of unsupervised methods that each aim to characterize different kinds of structure in the data. For instance, **cluster analysis** is an unsupervised learning method where the goal is to identify groups, or clusters, of statistically similar observations (Jain et al., 1999). **Collaborative filtering** seeks to "complete" a partial array of data, by leveraging correlations between data variables (Su and Khoshgoftaar, 2009). **Dimensionality reduction** posits that many datasets exhibit substantial redundancy across variables, and aims to reduce the data to its essential directions of variability (van der Maaten et al., 2009).



(a) Clustering (b) Dimensionality reduction (c) Density estimation

Figure 2.1: Unsupervised learning methods include clustering (a) where data is separated in statistically similar groups, (b) dimensionality reduction where the aim is to capture low-dimensional structure of the data, and (c) density estimation, where the aim is approximate the true data distribution.

Dimensionality reduction is closely related to **representation learning** in which the aim is to learn transformations of data that serve as useful representations for down-stream tasks (Bengio et al., 2013).

Many unsupervised learning approaches can be understood from a probabilistic perspective, where the goal is to find a model p_θ that closely matches the observed data. When dealing with continuous data this task is often referred to as **density estimation**. However, one issue with framing unsupervised learning in this way is that successful density estimation does not always incentivize the learning of useful structure in the data. For instance, consider a perfect black-box model that outputs calibrated probability densities for any input $\mathbf{x}$. Such a model has perfectly characterized the statistical dependencies between data variables, but it may not be useful to us if we are interested in cluster-structure, or low-dimensional representations of data for use in alternative tasks.

To reconcile the goals of unsupervised learning, with the generic probabilistic objective of density estimation we can impose *structure* on the parametric model p_θ. In doing so we obtain probabilistic analogs to a number of classical unsupervised objectives. For instance, if we assume the data are generated using unobserved latent variables $\mathbf{z}$, and that the latent variables are of lower dimensionality than the observed variables, then by performing inference we also do dimensionality reduction. Under certain modeling assumptions that are discussed in the following section, this reduces to a probabilistic variant of the classic dimensionality reduc-

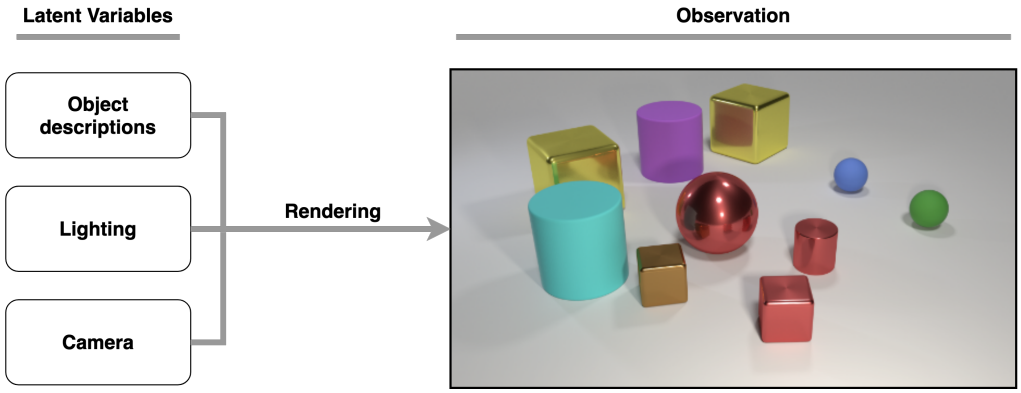


Figure 2.2: The data generating process for images from the CLEVR dataset (Johnson et al., 2017). To generate an image, latent scene variables (left) including object properties and lighting conditions are chosen from a prior model, and then transformed through the rendering process to a photo-realistic image (right).

tion method PCA (Tipping and Bishop, 1999). Alternatively if the latent variables are categorical cluster indicators, as in a mixture model, then we naturally recover a clustering method through the EM-algorithm (See Sec 2.2). This unifying probabilistic view of unsupervised learning is one that we subscribe to in this thesis, and all of our problems are formulated in probabilistic terms.

2.2 Latent variable models

Latent variable models assume that data variables $\mathbf{x}$ are generated via interactions with unobserved, or *latent* variables, typically denoted $\mathbf{z}$. Intuitively, it is reasonable to believe that in a particular dataset we will not have observed all the relevant variables, and that correlations between variables may be caused by some unobserved source. For example, if we collect data about umbrella sales and car accidents over time, we will probably observe positive correlations in the variables. However, these variables are really *independent*, given the knowledge that it is raining, or not raining, which in this case is a latent variable. Alternatively, for perceptual data like images of faces, we know that there exist some underlying factors that explain most of the variability across images: skin tone, face shape, camera pose, facial expression, etc. We expect the dimensionality of these latent factors to be smaller than the number of pixels in an image. Latent variable models formalize these assumptions by describing a data-generating process where

latent variables are first sampled, and then data variables are generated conditioned on these latent variables. For instance, to generate an image of a person we first generate the latent features $\mathbf{z}$ such as skin tone or camera pose, and then transform these features to pixels $\mathbf{x}$ via a digital image-formation process. Figure 2.2 depicts a similar generative process for images in the CLEVR dataset, with scene attributes such as object types, colours and materials being fed into a renderer to produce photo-realistic images (Johnson et al., 2017).

Latent variable models are useful ways to describe natural observations, and we can obtain the model distribution over the observed variables by marginalizing out the latent variables:

$$p(\mathbf{x};\boldsymbol{\theta}) = \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta})p(\mathbf{z};\boldsymbol{\theta})\ d\mathbf{z}. \tag{2.1}$$

This means that to evaluate the probability of a given observation, we need to consider all possible settings of the latent variables, weight them by their prior probabilities, and evaluate the probability of the observation assuming those particular latent variables.

2.2.1 Linear subspace models

A popular class of latent variable models for continuous data are linear-subspace models (Roweis and Ghahramani, 1999). These models are characterized by the assumption that the data $\mathbf{x} \in \mathbb{R}^D$ is generated by *linearly* transforming latent variables $\mathbf{z} \in \mathbb{R}^L$ using a weighting matrix $\mathbf{W} \in \mathbb{R}^{D\times L}$ and then adding a mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and uncorrelated noise $\boldsymbol{\epsilon} \in \mathbb{R}^D$. The model can be written

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}, \tag{2.2}$$

$$p(\mathbf{z};\boldsymbol{\theta}) = \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I}), \tag{2.3}$$

$$p(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{\epsilon} \mid \mathbf{0}, \boldsymbol{\Psi}), \tag{2.4}$$

where $\boldsymbol{\Psi} \in \mathbb{R}^{D\times D}$ is a diagonal covariance matrix. Using the fact that the product of Gaussian pdfs is Gaussian, and that the marginals of multivariate Gaussian distributions are Gaussian, it follows that the joint distribution $p(\mathbf{x}, \mathbf{z};\boldsymbol{\theta})$ and the marginal $p(\mathbf{x};\boldsymbol{\theta})$ are also Gaussian. We can obtain the parameters of the marginal

data distribution as follows:

$$\begin{aligned}
\mathbb{E}[\mathbf{x}] &= \mathbb{E}[\mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}] && (2.5)\\
&= \mathbf{W}\mathbb{E}[\mathbf{z}] + \boldsymbol{\mu} + \mathbb{E}[\boldsymbol{\epsilon}] && (2.6)\\
&= \boldsymbol{\mu}. && (2.7)\\
\mathrm{Cov}[\mathbf{x}] &= \mathrm{Cov}[\mathbf{x} - \boldsymbol{\mu}] && (2.8)\\
&= \mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] && (2.9)\\
&= \mathbb{E}[(\mathbf{W}\mathbf{z} + \boldsymbol{\epsilon})(\mathbf{W}\mathbf{z} + \boldsymbol{\epsilon})^T] && (2.10)\\
&= \mathbf{W}\mathbb{E}[\mathbf{z}\mathbf{z}^T]\mathbf{W}^T + \mathbf{W}\mathbb{E}[\mathbf{z}\boldsymbol{\epsilon}^T] + \mathbb{E}[\boldsymbol{\epsilon}\mathbf{z}^T]\mathbf{W}^T + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T] && (2.11)\\
&= \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}. && (2.12)
\end{aligned}$$

As such linear-subspace models are simply multivariate Gaussians with covariances that are restricted to be the sum of a low-rank matrix $\mathbf{W}\mathbf{W}^T$, and a diagonal matrix $\boldsymbol{\Psi}$. Depending on the form of the diagonal matrix $\boldsymbol{\Psi}$, different models can be obtained:

Probabilistic principal components analysis. If we assume that the noise variance is equal in each dimension $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$ then we obtain probabilistic principal components analysis (PPCA, Tipping and Bishop, 1999). A maximum-likelihood solution for the parameters $\mathbf{W}, \boldsymbol{\mu}, \sigma^2$ are given as follows:

$$\hat{\mathbf{W}} = \mathbf{V}(\boldsymbol{\Lambda} - \hat{\sigma^2}\mathbf{I}), \tag{2.13}$$

$$\hat{\boldsymbol{\mu}} = \frac{\sum_{n=1}^{N} \mathbf{x}_n}{N}, \tag{2.14}$$

$$\hat{\sigma^2} = \frac{1}{D-L}\sum_{j=L+1}^{D} \lambda_j, \tag{2.15}$$

where the columns of $\mathbf{V}$ are the eigenvectors of the empirical data covariance matrix, and $\boldsymbol{\Lambda}$ is a diagonal matrix with the corresponding eigenvalues $[\lambda_1, \lambda_2, \ldots, \lambda_D]$ as the diagonal entries. The connection to PCA is apparent from the maximum-likelihood solutions: The estimated projection matrix $\hat{\mathbf{W}}$ is constructed using the PCA eigenvectors and eigenvalues, and as $\sigma \to 0$ we obtain exactly the PCA eigenvector matrices scaled by their corresponding eigenvalues.

Factor analysis. If we assume that $\boldsymbol{\Psi}$ is an arbitrary diagonal matrix with positive entries on the diagonal, then we obtain the factor analysis (FA) model (Rubin and Thayer, 1982). Unlike PPCA, it is not possible to obtain the maximum-

likelihood solution for the model parameters analytically, and numerical methods must be used as described in the following section.

Specifying a full covariance matrix requires $D(D+1)$ / 2 parameters, which can be problematic when D is large relative to the number of data examples N. PPCA and FA reduce the number of parameters required to specify the model covariance matrix to $LD+1$ and $LD+D$ respectively, and so they are useful in controlling model capacity in scenarios where not enough data is available to effectively constrain a full covariance matrix.

2.2.2 Exact inference and the EM algorithm

In the PPCA model it is possible to obtain a maximum-likelihood estimate of the model parameters analytically. However, for many latent variable models there is no analytic maximum-likelihood solution, and so numerical methods must be used to estimate model parameters. The goal of such methods is to maximize the log-probability of the observed data. For latent variable models this has the form:

$$l(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}_n; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log \int_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta}) d\mathbf{z}_n. \tag{2.16}$$

For models where integration of the latent variables is tractable, and where $\log p(\mathbf{x}; \boldsymbol{\theta})$ is differentiable with respect to $\boldsymbol{\theta}$, we can apply gradient-based methods to optimize $l(\boldsymbol{\theta})$ directly. We can also use a popular bound-based optimization approach called **expectation-maximization** (**EM**). The EM-algorithm is motivated by the following observation: If we knew the values of the latent variables associated with a given observation then we could obtain parameter estimates by maximizing the **complete-data log-likelihood**:

$$\mathcal{C}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta}). \tag{2.17}$$

While we don't know the values of the latent variables required for evaluating $\mathcal{C}(\boldsymbol{\theta})$, for our current best model of the data, we can invert our current best model of the data using Bayes' rule to obtain the posterior $p(\mathbf{z}|\mathbf{x}; \boldsymbol{\theta}_c)$, where $\boldsymbol{\theta}_c$ are the parameters of the current model. The posterior captures our current model's beliefs about the latent variables associated with $\mathbf{x}$. This enables us to marginalize out the latent variables in $\mathcal{C}(\boldsymbol{\theta})$ while weighting by the posterior beliefs. The

resulting expression is known as the **expected complete-data log-likelihood**:

$$\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c) = \sum_{n=1}^{N} \int_{\mathbf{z}_n} p(\mathbf{z}_n|\mathbf{x}_n;\boldsymbol{\theta}_c) \log p(\mathbf{x}_n,\mathbf{z}_n;\boldsymbol{\theta}) d\mathbf{z}_n \tag{2.18}$$

$$= \sum_{n=1}^{N} \mathbb{E}_{p(\mathbf{z}_n|\mathbf{x}_n;\boldsymbol{\theta}_c)} \left[\log p(\mathbf{x}_n,\mathbf{z}_n;\boldsymbol{\theta})\right]. \tag{2.19}$$

This expression no longer depends on the unobserved latents, and it can therefore be maximized with respect to the $\boldsymbol{\theta}$ while keeping $\boldsymbol{\theta}_c$ fixed:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c). \tag{2.20}$$

For many latent variable models $l(\boldsymbol{\theta})$ can't be maximized analytically, while $\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c)$ can. There is a problem here, which is that the posterior over $\mathbf{z}$ is highly dependent on the parameters $\boldsymbol{\theta}$. The EM-algorithm addresses this problem by repeatedly alternating between taking posterior *expectations* (**E-step**) and *maximization* of the expected complete-data log-likelihood (**M-step**). The M-step improves the model, which in turn leads to better posterior predictions in the E-step. In practice only certain terms of $\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c)$ depend on the parameters $\boldsymbol{\theta}$, and the E-step can be reduced to evaluating these terms, which are known as the *expected sufficient statistics*.

1. **E-step.** Evaluate sufficient statistics of $\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c)$ as required for M-step.
2. **M-step.** Optimize $\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c)$ with respect to $\boldsymbol{\theta}$ using the E-step sufficient statistics.

EM-updates for the FA model. For the E-step we first derive the form of the posterior $p(\mathbf{z}|\mathbf{x};\boldsymbol{\theta})$. As $\mathbf{x}$ and $\mathbf{z}$ are jointly Gaussian we can obtain the posterior using standard results for Gaussian conditionals:

$$p(\mathbf{z}|\mathbf{x};\boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}|\mathbf{m},\boldsymbol{\Sigma}), \tag{2.21}$$

$$\boldsymbol{\Sigma} = (\mathbf{I} + \mathbf{W}^T\boldsymbol{\Psi}^{-1}\mathbf{W})^{-1}, \tag{2.22}$$

$$\mathbf{m} = \boldsymbol{\Sigma}\left(\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right). \tag{2.23}$$

The posterior mean is therefore a linear function of the centered data variables, and the posterior covariance does not depend on the data variables at all. Given the posterior, the next step is to obtain the M-step updates, and the sufficient statistics required from the E-step by optimizing the expected complete-data log

likelihood. Starting with the complete data log likelihood and excluding constants we have

$$\mathcal{C}(\boldsymbol{\theta}) = -\frac{1}{2}\sum_{n=1}^{N} \log|\boldsymbol{\Psi}| + (\mathbf{x}_n - \mathbf{W}\mathbf{z}_n)^T\boldsymbol{\Psi}^{-1}(\mathbf{x}_n - \mathbf{W}\mathbf{z}_n) \tag{2.24}$$

$$= -\frac{1}{2}\sum_{n=1}^{N} \log|\boldsymbol{\Psi}| + \mathrm{tr}\left[(\mathbf{x}_n - \mathbf{W}\mathbf{z}_n)(\mathbf{x}_n - \mathbf{W}\mathbf{z}_n)^T\boldsymbol{\Psi}^{-1}\right]. \tag{2.25}$$

Taking expectations with respect to $p(\mathbf{z}|\mathbf{x};\boldsymbol{\theta})$ and using the linearity of the trace operator we obtain the expected complete-data log-likelihood:

$$\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c) = -\frac{1}{2}\sum_{n=1}^{N} \log|\boldsymbol{\Psi}| + \mathrm{tr}[\mathbf{x}_n\mathbf{x}_n^T\boldsymbol{\Psi}^{-1}] - 2\mathrm{tr}[\mathbb{E}[\mathbf{z}_n]^T\mathbf{W}^T\mathbf{x}_n\boldsymbol{\Psi}^{-1}] \tag{2.26}$$

$$+ \mathrm{tr}\left[\mathbf{W}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^T]\mathbf{W}^T\boldsymbol{\Psi}^{-1}\right]. \tag{2.27}$$

Taking derivatives with respect to $\mathbf{W}$ and $\boldsymbol{\Psi}$ we obtain:

$$\frac{\delta\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c)}{\delta\mathbf{W}} = -\boldsymbol{\Psi}^{-1}\frac{1}{2}\sum_{n=1}^{N}\mathbf{x}_n\mathbb{E}[\mathbf{z}_n]^T + \boldsymbol{\Psi}^{-1}\frac{1}{2}\mathbf{W}\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^T], \tag{2.28}$$

$$\frac{\delta\mathcal{Q}(\boldsymbol{\theta};\boldsymbol{\theta}_c)}{\delta\boldsymbol{\Psi}^{-1}} = \frac{N}{2}\boldsymbol{\Psi} - \frac{1}{2}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^T - 2\mathbb{E}[\mathbf{z}_n]^T\mathbf{W}^T\mathbf{x}_n + \mathbf{W}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^T]\mathbf{W}^T. \tag{2.29}$$

Finally, by setting the derivatives to zero and solving we obtain the M-step analytic updates:

$$\hat{\mathbf{W}} = \left(\sum_{n=1}^{N}\mathbf{x}_n\mathbb{E}[\mathbf{z}_n]^T\right)\left(\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^T]\right)^{-1}, \tag{2.30}$$

$$\hat{\boldsymbol{\Psi}} = \frac{1}{N}\mathrm{diag}\left(\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\mathbf{W}}\mathbb{E}[\mathbf{z}_n])\mathbf{x}_n^T\right). \tag{2.31}$$

From these equations we can observe the required expected sufficient statistics: $\mathbb{E}[\mathbf{z}_n]$ and $\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^T]$ for $n = 1, \ldots, N$ which can be determined using and the mean $\mathbf{m}$ and covariance $\boldsymbol{\Sigma}$ of $p(\mathbf{z}|\mathbf{x};\boldsymbol{\theta})$:

$$\mathbb{E}[\mathbf{z}_n] = \mathbf{m}_n = \boldsymbol{\Sigma}\left(\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x}_n - \boldsymbol{\mu})\right), \tag{2.32}$$

$$\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^T] = \mathrm{Cov}(\mathbf{z}_n|\mathbf{x}_n) + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^T = \boldsymbol{\Sigma} + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^T. \tag{2.33}$$

For FA the E-step is efficient, as $\boldsymbol{\Sigma}$ can be computed once per E-step, and the posterior means for each input example can be evaluated in parallel with a single matrix multiplication. The most expensive part is inverting a potentially high-dimensional matrix for the covariance, but this cost can be reduced by making use

of the low-rank structure using the Woodbury identity (Woodbury and Woodbury, 1950).

Why does the EM algorithm work? One way to justify the use of the EM-algorithm is to show that it maximizes a lower-bound on the model log-likelihood. In the E-step we take expectations of the complete data log-likelihood with respect to the model posterior $p(\mathbf{z}|\mathbf{x};\boldsymbol{\theta})$, but it may not be obvious that this is this the best choice of distribution to use. Instead, let $q(\mathbf{z})$ be an arbitrary distribution over the latent variables. We obtain the following lower bound on the model log-likelihood using Jensen's equality:

$$
\begin{aligned}
l(\boldsymbol{\theta}) &= \sum_{n=1}^{N} \log \int_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta}) d\mathbf{z}_n && (2.34)\\
&= \sum_{n=1}^{N} \log \int_{\mathbf{z}_n} q(\mathbf{z}_n) \frac{p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta})}{q(\mathbf{z}_n)} d\mathbf{z}_n && (2.35)\\
&\geq \sum_{n=1}^{N} \int_{\mathbf{z}_n} q(\mathbf{z}_n) \log \frac{p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta})}{q(\mathbf{z}_n)} d\mathbf{z}_n && (2.36)\\
&= \sum_{n=1}^{N} \mathbb{E}_{q(\mathbf{z}_n)} \left[\log \frac{p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta})}{q(\mathbf{z}_n)} \right] = \mathcal{L}(\boldsymbol{\theta}). && (2.37)
\end{aligned}
$$

$\mathcal{L}(\boldsymbol{\theta})$ is often referred to as the *evidence lower bound*, or the *variational lower bound*. It can be decomposed as follows:

$$
\begin{aligned}
\mathcal{L}_n(\boldsymbol{\theta}) &= \mathbb{E}_{q(\mathbf{z}_n)} \left[\log \frac{p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta})}{q(\mathbf{z}_n)} \right] && (2.38)\\
&= \mathbb{E}_{q(\mathbf{z}_n)} \left[\log \frac{p(\mathbf{z}_n|\mathbf{x}_n; \boldsymbol{\theta})}{q(\mathbf{z}_n)} \right] + \mathbb{E}_{q(\mathbf{z}_n)} \left[\log p(\mathbf{x}_n; \boldsymbol{\theta}) \right] && (2.39)\\
&= -\mathbb{D}_{\mathrm{KL}}[q(\mathbf{z}_n)||p(\mathbf{z}_n|\mathbf{x}_n; \boldsymbol{\theta})] + \log p(\mathbf{x}_n; \boldsymbol{\theta}), && (2.40)
\end{aligned}
$$

where $D_{\mathrm{KL}}[q(\mathbf{z}_n) \,||\, p(\mathbf{z}_n|\mathbf{x}_n;\boldsymbol{\theta})]$ is the KL-divergence between the two distributions. The difference between the model log-probability and the variational lower bound is exactly the KL divergence between $q(\mathbf{z}_n)$ and the true posterior. The lower bound can therefore be maximised by *minimizing* the KL-divergence with respect to $q(\mathbf{z}_n)$, and the best that can be done is to set $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n;\boldsymbol{\theta})$, reducing the KL-divergence to zero. Therefore the E-step maximizes the lower bound over all possible distributions q.

For the M-step consider an alternative decomposition of the ELBO:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathbb{E}_{q(\mathbf{z}_n)} \left[\log \frac{p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta})}{q(\mathbf{z}_n)} \right] \tag{2.41}$$

$$= \sum_{n=1}^{N} \mathbb{E}_{q(\mathbf{z}_n)} [\log p(\mathbf{x}_n, \mathbf{z}_n; \boldsymbol{\theta})] + \mathcal{H}(q(\mathbf{z}_n)), \tag{2.42}$$

where $\mathcal{H}(q(\mathbf{z}_n))$ is the entropy of $q(\mathbf{z}_n)$. The first term is the expected complete data log-likelihood where expectations are taken with respect to q. By performing the E-step and setting $q(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{x}_n; \boldsymbol{\theta})$, the first term is exactly the complete data log-likelihood $\mathcal{Q}(\boldsymbol{\theta}; \boldsymbol{\theta}_c)$, and as the second term does not contain any terms involving $\boldsymbol{\theta}$, so maximization of $\mathcal{Q}(\boldsymbol{\theta}; \boldsymbol{\theta}_c)$ maximizes the lower bound with respect to the model parameters $\boldsymbol{\theta}$.

Gradient descent vs the EM-algorithm As mentioned at the start of this section, for some classes of model it is possible to optimize the model parameters by integrating out the latent variables, and using gradient descent to directly maximize the log-likelihood. So should we use EM or gradient descent to fit these models? Salakhutdinov et al. (2003) show that gradient descent can demonstrate poor convergence properties relative to alternatives, at least for relatively small models trained using standard gradient descent. The picture is more nuanced, as it is now typical to train large models on large datasets, using modified stochastic first order methods, that update parameters using small batches of randomly-selected data examples (Goodfellow et al., 2016). In addition, models are commonly not trained to convergence, using 'early-stopping', to stop optimization before overfitting occurs (Goodfellow et al., 2016). Given these factors, we should be wary of avoiding first order methods based on the earlier results, and we note that gradient descent has been successfully used to train latent variable models for a range of applications (Bishop, 1994; Graves, 2013).

2.2.3 Variational inference for latent variable models

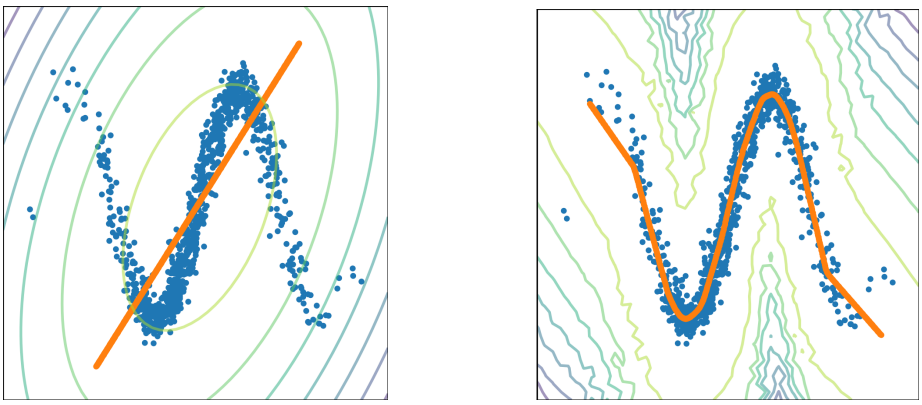
In the previous section we made use of the model posterior $p(\mathbf{z}|\mathbf{x}; \boldsymbol{\theta})$ as a key component in the EM algorithm. For the FA model the parameters of the posterior can be obtained analytically and relatively inexpensively, however, evaluating posteriors over latent variables is not always so straightforward. For instance, if the data has missing values, then the posterior covariance matrix for the FA

model depends on which variables are missing, and so a separate matrix inversion must be performed for each input example in the E-step, substantially increasing the overall cost of the EM-algorithm (See Chapter 4). In other cases, the only way to obtain the latent posterior is through an intractable integration. This makes application of the standard EM algorithm infeasible for many latent variable models.

Variational inference seeks to reduce the cost of inference by substituting the exact posterior with an approximate posterior that is easier to evaluate. When performing variational inference we typically choose a family of approximating distributions $q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})$[1] with parameters $\boldsymbol{\phi}$. Often a family that simplifies the inference process is chosen, for instance a factored, or mean-field distribution $q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi}) = \prod_d q(z_d|\mathbf{x};\boldsymbol{\phi})$ is a common choice in many algorithms (Saul et al., 1996; Blei et al., 2017). In the context of the EM-algorithm we replace the standard E-step with a variational E-step where instead of explicitly maximizing the variational lower-bound by choosing the exact posterior, it is maximized as far as possible with respect to $\boldsymbol{\phi}$. This is equivalent to minimizing the KL-divergence between the approximating distribution and the exact posterior $D_{\mathrm{KL}}[q(\mathbf{z}_n|\mathbf{x}_n;\boldsymbol{\phi})||p(\mathbf{z}_n|\mathbf{x}_n;\boldsymbol{\theta})]$. The extent to which this minimization is possible is determined by the family of approximating distributions. For the FA model for instance, a posterior approximation with diagonal covariance cannot match the true posterior which has full covariance. In many cases it is possible to analytically obtain the parameters $\boldsymbol{\phi}$ that maximize the ELBO for a particular variational family, and when it is not possible, numerical methods can be used. Note that unlike the standard EM-algorithm, for some variational families, variational EM will not converge to even a local maximum of the likelihood.

Variational EM-updates for the FA model. As an example we derive the variational E-step updates for the FA model with a mean-field approximating distribution. Let $q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi}) = \mathcal{N}(\mathbf{z}|\mathbf{m}_{\boldsymbol{\phi}}, \boldsymbol{\Sigma}_{\boldsymbol{\phi}})$ where $\boldsymbol{\Sigma}_{\boldsymbol{\phi}}$ is a diagonal matrix, and as before let $p(\mathbf{z}|\mathbf{x};\boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}|\mathbf{m}, \boldsymbol{\Sigma})$. The KL-divergence between the approximate

[1] In contrast to the previous section where we used q to denote an arbitrary variational distribution.



(a) Factor Analysis (b) Deep latent variable model

Figure 2.3: (a) Factor analysis and (b) Deep latent variable model with 1-dimensional latent spaces fitted to a noisy sine-wave dataset. The orange lines indicate the decoded latent manifold from $z = [-3, 3]$. Contours of the models' log probability density are shown.

and true distributions is:

$$D_{\mathrm{KL}}[q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})||p(\mathbf{x}|\mathbf{x};\boldsymbol{\phi})] = \int_{\mathbf{z}} q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi}) \log \frac{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}{p(\mathbf{z}|\mathbf{x};\boldsymbol{\theta})} d\mathbf{z} \tag{2.43}$$

$$= \frac{1}{2}\left[\log \frac{|\boldsymbol{\Sigma}|}{|\boldsymbol{\Sigma}_{\boldsymbol{\phi}}|} - d + \mathrm{Tr}(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{\phi}}) + (\mathbf{m} - \mathbf{m}_{\boldsymbol{\phi}})\boldsymbol{\Sigma}^{-1}(\mathbf{m} - \mathbf{m}_{\boldsymbol{\phi}})\right]. \tag{2.44}$$

By taking gradients with respect to $\mathbf{m}_{\boldsymbol{\phi}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\phi}}$ and solving we obtain

$$\boldsymbol{\Sigma}_{\boldsymbol{\phi}} = \left[\mathrm{diag}\left(\boldsymbol{\Sigma}^{-1}\right)\right]^{-1}, \tag{2.45}$$

$$\mathbf{m}_{\boldsymbol{\phi}} = \mathbf{m}, \tag{2.46}$$

which is simply the exact posterior, but with only the diagonal elements of the covariance matrix. In this case, the exact variational approximation is not cheaper than simply using the exact posterior, as evaluating $\mathbf{m}$ involves inverting the covariance of the exact posterior.

2.3 Deep latent variable models

Linear subspace models enable us to model high-dimensional data by making assumptions about underlying low-dimensional structure. Recall that for linear subspace models the mean of the conditional data generating distribution is a

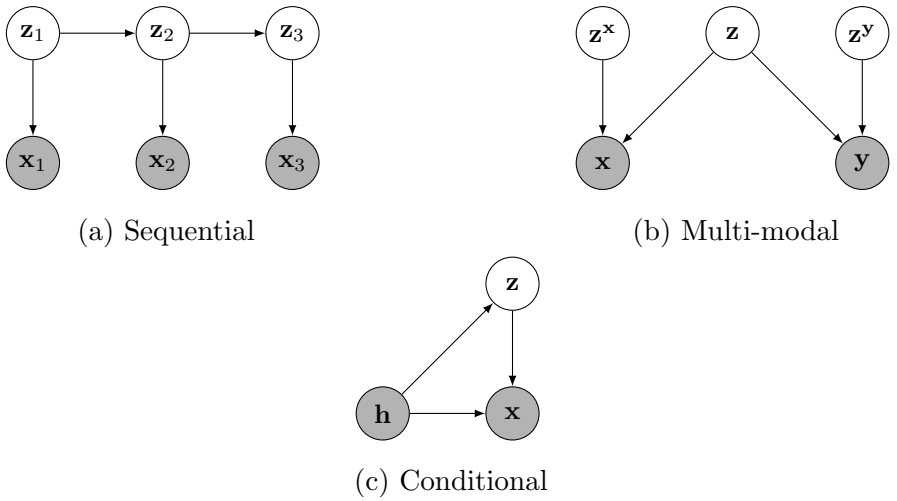
linear function of the latent variables, and the covariance does not depend on the latent variables (See Equation 2.2). As illustrated in Figure 2.3(a), for data with complex dependencies, this linear transformation can be too restrictive, and the model is unable to effectively fit the data. But with the advent of deep learning, we now have access to an increasingly mature toolkit for the efficient approximation of complex non-linear functions. Deep learning represents functions using neural networks that successively transform input data towards the desired outputs. It is a vast field, and is responsible for substantial improvements in capability in a wide range of domains, from image recognition, machine translation. A full review is beyond the scope of this thesis, but we can think of deep learning as providing an increasingly mature toolkit for representing generic functions. We refer readers to LeCun et al. (2015) for a summary of deep learning advances, and to Goodfellow et al. (2016) for textbook material. Deep latent variable models take advantage of the toolkit provided by deep learning by using neural networks that directly transform latent variables into output parameters. This enables us to represent complex data distributions as shown in Figure 2.3(b).

Let $\boldsymbol{\pi}(\mathbf{z};\boldsymbol{\theta})$ represent the parameters of the conditional data distribution obtained by passing latent variables $\mathbf{z}$ through a neural network with parameters $\boldsymbol{\theta}$. The data generation process can therefore be defined as:

$$\mathbf{z} \sim p(\mathbf{z};\boldsymbol{\theta}), \tag{2.47}$$

$$\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\pi}(\mathbf{z};\boldsymbol{\theta})). \tag{2.48}$$

In this thesis we typically omit the explicit dependence on the parameters $\boldsymbol{\pi}$, and denote the conditional data distribution as $p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta})$. The network that transforms the latent variables is often referred to as the generator network, or decoder. Typically the prior is chosen to be a simple, fixed model such as a spherical Gaussian $p(\mathbf{z};\boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}|\mathbf{0},\mathbf{I})$, although it is possible to use a parameterized distribution that is trained jointly with the decoder. For continuous data, a common choice of conditional data distribution is to use a factorized Gaussian, with means and variances that depend on the latent variables: $p(\mathbf{x}|\boldsymbol{\pi}(\mathbf{z};\boldsymbol{\theta})) = \prod_d \mathcal{N}(x_d|\mu_d(\mathbf{z};\boldsymbol{\theta}),\sigma_d^2(\mathbf{z};\boldsymbol{\theta}))$. For binary data the generator network typically outputs Bernoulli probabilities $p(\mathbf{x}|\boldsymbol{\pi}(\mathbf{z};\boldsymbol{\theta})) = \prod_d \mathcal{B}(x_d|\pi_d(\mathbf{z};\boldsymbol{\theta}))$. In either case, it is typical to use distributions that are conditionally independent across data variables, given the latent variables. This conditional independence is significant, as it carries the strong assumption that all the dependencies in the data can be



(a) Sequential

(b) Multi-modal

(c) Conditional

Figure 2.4: Example structured latent variable models with visible and latent variables indicated as shaded and blank circles respectively. (a) Sequential models of time-series data where the time-dependence of the visible variables is explained by time-dependent latents. (b) Multi-modal models capture distributions over distinct data-modalities through the use of both shared and mode-specific latent variables. (c) Conditional models can use latent variables to explain additional dependencies in the data.

explained by a collection of low-dimensional latent variables.

For the neural network components, it is common to use fully-connected networks to model tabular data, and convolutional networks to model image data, although any differentiable network components can be used. As with ordinary latent variable models, deep models can be granted additional structure in order to better model data from certain modalities, such as recurrent latent connections for time-series models, or shared latent variables for multi-modal modelling (Figure 2.4).

2.3.1 Conditional generative models

Often we are interested in modeling not just the values of some random variable $\mathbf{x}$, but the values of $\mathbf{x}$ when conditioned on some context $\mathbf{h}$. Consider for instance the task of modeling natural images given an image class label (van den Oord et al., 2016b), or of modeling sentences in French conditioned on sentences in English (Sutskever et al., 2014), or even of modelling raw audio waves, conditioned on a speaker's words and identity (van den Oord et al., 2016a). These can all be

framed as *conditional* generative modelling tasks, where the aim is to describe a conditional distribution $p(\mathbf{x}|\mathbf{h})$. In each of these examples the output variables are not fully determined by the conditioning information. For instance, for a class label such as "horse", there are many coherent images that match this description. Therefore it is important to characterize the variability and dependencies in $\mathbf{x}$ that are not captured by $\mathbf{h}$. Conditional generative models aim to capture this variability through estimation of $p(\mathbf{x}|\mathbf{h})$. We distinguish between conditional generative modelling and classic supervised learning tasks like image classification by the high-dimensionality and complexity of the conditional distribution in the former.

One way of modeling these dependencies is to use latent variables as follows:

$$p(\mathbf{x}|\mathbf{h};\boldsymbol{\theta}) = \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{h},\mathbf{z};\boldsymbol{\theta})p(\mathbf{z}|\mathbf{h};\boldsymbol{\theta})d\mathbf{z}. \tag{2.49}$$

For example the latent variables $\mathbf{z}$ could capture lighting, pose, or many other kinds of variability present in the horse-labelled images. Here both the latent variable distribution and observation model condition on the context, but models where the latent variables are independent of the context may also be of interest. As with the unconditional deep latent variable models described in the previous section, conditional latent variable models can make use of neural network components in order to express complex relationships between $\mathbf{h}$, $\mathbf{z}$ and $\mathbf{x}$ (Sohn et al., 2015). Figure 2.4(c) shows a graphical representation of conditional latent variable models. In Chapter 3 we demonstrate an approach to modeling 3D shape variability using conditional deep generative models.

2.3.2 Amortized variational inference

For the linear-subspace models described in Section 2.2.1 it is possible to analytically obtain the posterior distribution over $\mathbf{z}$ in an efficient way and to therefore train using the standard EM-algorithm. This is a consequence of the fact that for such models the joint distribution of latents and data variables is Gaussian, and that conditional subsets of jointly Gaussian variables are also Gaussian. For deep latent variables that is no longer the case, and solving the integral $\int_{\mathbf{z}} p(\mathbf{x},\mathbf{z};\boldsymbol{\theta})d\mathbf{z}$ required to evaluate the posterior is intractable. Directly optimizing the model likelihood with gradient-based methods is also not possible for the same reason. As such, approximate inference methods are required to train these models.

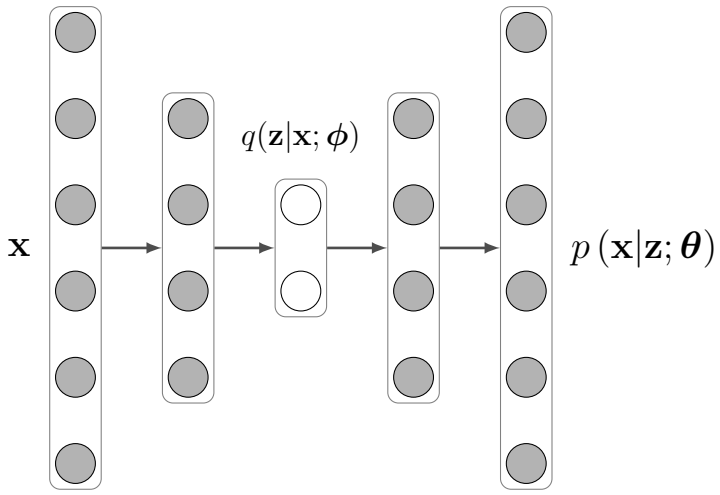


Figure 2.5: Variational autoencoder. Inputs $\mathbf{x}$ are passed through an encoder to obtain the parameters of an approximate posterior $q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})$. A sample $\mathbf{z}$ from this distribution is passed through a decoder which outputs the parameters $\boldsymbol{\theta}$ of the conditional data distribution $p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta})$.

Monte-Carlo EM (Wei and Tanner, 1990), where the expected complete-data log likelihood Q is approximated using MCMC samples, is one such method. But posterior sampling using MCMC is infeasible given the expense of passing latent variables through a generator network, the typically poor scaling of MCMC methods with the dimensionality of the latent variables, and the need to do posterior inference for each data example. Classic variational methods (Section 2.2.3) are an alternative, but it is not possible to derive analytic mean-field updates, and per-data point numerical optimization of a variational distribution is expensive.

The solution to these challenges proposed in Kingma and Welling (2014) and Rezende et al. (2014), is one of the key ideas underlying modern deep latent variable models. It is based on the observation that there exists a mapping between input data points and the optimal variational parameters, and that this mapping can itself be approximated with a neural network. The cost of variational optimization can therefore be *amortized* across different examples using a **recognition network** that takes as input data variables $\mathbf{x}$ and outputs the parameters of an approximate posterior. One of the key insights is that the approximate posterior can be trained jointly with the generative model by optimizing the lower-bound $\mathcal{L}$ with respect to parameters of both networks using first-order gradient methods. Recall from section 2.2.2 the following variational

lower bound:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}\left[\log \frac{p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})}\right] \tag{2.50}$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}\left[\log p(\mathbf{x}|\mathbf{z}; \boldsymbol{\theta}) + \log p(\mathbf{z}; \boldsymbol{\theta}) - \log q(\mathbf{z}|\mathbf{x}; \boldsymbol{\theta})\right]. \tag{2.51}$$

One way to obtain an estimate of the gradients of the lower-bound with respect to the parameters of the approximate posterior is using an S-sample estimate of the *score-function*:

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}\left[\log \frac{p(\mathbf{x}, \mathbf{z}; \boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})} \nabla_{\boldsymbol{\phi}} \log q(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})\right] \tag{2.52}$$

$$\simeq \frac{1}{S} \sum_{s=1}^{S} \log \frac{p(\mathbf{x}, \mathbf{z}^{(s)}; \boldsymbol{\theta})}{q(\mathbf{z}^{(s)}|\mathbf{x}; \boldsymbol{\phi})} \nabla_{\boldsymbol{\phi}} \log q(\mathbf{z}^{(s)}|\mathbf{x}; \boldsymbol{\phi}), \tag{2.53}$$

$$\mathbf{z}^{(s)} \sim q(\mathbf{z}^{(s)}|\mathbf{x}; \boldsymbol{\phi}). \tag{2.54}$$

Alternatively a low-variance estimator of the expectation can be obtained by reparameterizing the approximate posterior so that samples can be obtained by transforming samples from a base distribution:

$$\mathbf{z} = \mathbf{f}(\mathbf{x}, \boldsymbol{\epsilon}), \tag{2.55}$$

$$\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}), \tag{2.56}$$

where $\mathbf{f}$ is a differentiable function of $\mathbf{x}$ and $\boldsymbol{\epsilon}$. For instance if $q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{z}|\mu(\mathbf{x}), \sigma^2(\mathbf{x})\mathbf{I})$ then samples can be obtained as:

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I}), \tag{2.57}$$

$$f(\boldsymbol{\epsilon}, \mathbf{x}) = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\sigma}(\mathbf{x}) \odot \boldsymbol{\epsilon}. \tag{2.58}$$

This results in the following low-variance estimator of the ELBO

$$\widetilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \frac{1}{S} \sum_{s=1}^{S} \log p(\mathbf{x}|\mathbf{z}^{(s)}; \boldsymbol{\theta}) + \log p(\mathbf{z}^{(s)}; \boldsymbol{\theta}) - \log q(\mathbf{z}^{(s)}|\mathbf{x}; \boldsymbol{\theta}), \tag{2.59}$$

$$\mathbf{z}^{(s)} = \mathbf{f}(\mathbf{x}, \boldsymbol{\epsilon}^{(s)}), \quad \boldsymbol{\epsilon}^{(s)} \sim p(\boldsymbol{\epsilon}). \tag{2.60}$$

By taking gradients with respect to the inference network and generator networks $\nabla_{\theta,\phi} \widetilde{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$ we can optimize the lower bound with first-order methods. This method is known as the reparameterization trick, which comes with the caveat that the samples from the posterior family must be reparameterizable using a base distribution and a differentiable transformation.

The variational lower bound is often decomposed into two terms: the reconstruction log probability $\log p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta})$ and the KL-divergence between the posterior and prior $D_{\mathrm{KL}}[q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})||p(\mathbf{z};\boldsymbol{\theta})]$

$$\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}\left[\log p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta}) + \log p(\mathbf{z};\boldsymbol{\theta}) - \log q(\mathbf{z}|\mathbf{x};\boldsymbol{\theta})\right] \tag{2.61}$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}\left[\log p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta})\right] - \mathbb{E}_{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}\left[\log\left(\frac{q(\mathbf{z}|\mathbf{x};\boldsymbol{\theta})}{p(\mathbf{z};\boldsymbol{\theta})}\right)\right] \tag{2.62}$$

$$= \mathbb{E}_{q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})}\left[\log p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta})\right] - D_{\mathrm{KL}}[q(\mathbf{z}|\mathbf{x};\boldsymbol{\phi})||p(\mathbf{z};\boldsymbol{\theta})]. \tag{2.63}$$

In some cases the KL-divergence term can be evaluated analytically and cheaply, e.g. for Gaussian priors and posteriors with diagonal covariance matrices.

2.3.3 Variational Autoencoder

Deep latent variable models that use amortized variational inference with a recognition network are known broadly as variational autoencoders (VAEs). This is due to the resemblance in the training procedure to classical autoencoders.

In regular autoencoders, inputs are transformed to a collection of hidden units using an encoder network, and these hidden units are then decoded in order to reconstruct the inputs. Often regularization such as dropout, or L1 regularization, is applied to the hidden code, in order to limit the capacity of the bottleneck units and shape the representations learned by the network (Arpit et al., 2016). Similarly, when training a deep latent variable model using amortized variational inference, the inputs are passed through an approximate inference network to obtain a posterior over latent variables. A latent variable is sampled from this posterior, and is then passed through the generator network to obtain the parameters of the conditional data distribution. The approximate inference network and generator network resemble the encoder and decoder networks in the standard autoencoder, and the KL-divergence term in the variational lower bound has a regularizing effect on the latent units. The terminology of encoding and decoding is often used to refer to approximate inference and data generation respectively. Similar connections have been made between auto-encoding and EM-algorithm inference in the context of linear Gaussian latent variable models (Roweis and Ghahramani, 1999). Figure 2.5 shows the structure of a simple VAE.

It is useful to note that although the inference network is an important part

of the training procedure for a deep latent variable model, it is *not* part of the data-generating process. A VAE is not just a model of data, but a model of data fused with a particular inference strategy that makes training tractable. However in practice, the use of amortized and approximate variational inference has a substantial impact on the resulting generative model, and can negatively impact the expressive capacity of the resulting model. A substantial body of work has investigated techniques that can be used to improve these properties (Burda et al., 2016; Rainforth et al., 2018; Cremer et al., 2018).

2.4 Alternative deep generative models

One of the primary challenges in generative modeling is to design an expressive parametric distribution for high-dimensional variables. In this chapter we have focused so far on models that use latent variables to implicitly capture dependencies between data variables, that are trained through exact or approximate posterior inference of the latent variables. However, there are a number of alternative approaches to modeling high-dimensional data that each have their merits and use-cases.

2.4.1 Flow-based models

Flow-based models (normalizing flows) represent densities using a deterministic invertible transformation of a base density. They can be thought of as a special case of the deep generative models discussed in this chapter, where $p(\mathbf{x}|\mathbf{z};\boldsymbol{\theta}) = \delta\left(\mathbf{x} - \mathbf{T}_\theta(\mathbf{z})\right)$, $\mathbf{T}_\theta$ is a parameterized invertible transformation, and the dimensionality of $\mathbf{z}$ is equal to that of the data. This invertibility means we can avoid doing approximate inference over the latent variables, and can instead directly optimize the model likelihood through the change of variables formula:

$$p(\mathbf{x}) = p\left(\mathbf{T}_\theta^{-1}(\mathbf{x});\theta\right)\left|\det\, \mathbf{J}_{T_\theta^{-1}}(\mathbf{x})\right|. \tag{2.64}$$

However this exact training objective comes at a cost: it restricts us to invertible transformations of the latent variables for which the determinant of the Jacobian of the inverse transformation is efficient to compute. Flow-based models were first used for deep generative modelling in Dinh et al. (2017) and have been used to

improve inference in latent variable models (Rezende and Mohamed, 2015; Kingma et al., 2016) as well as density estimation (Papamakarios et al., 2017; Durkan et al., 2019). For a recent review see Papamakarios et al. (2019). Compared to the deep latent variable models that are the focus of this thesis, flows present architecture challenges when it comes to trying to explicitly capture certain kinds of structure in data. For instance, in Chapter 3 we use latent variables to capture low-dimensional structure in point-cloud data, and in Chapter 5 we use a set of latent vectors to represent distinct objects in visual scenes. It is not obvious how to design invertible transformations that would provide these structured representations, but it is an interesting direction for future work.

2.4.2 Generative adversarial networks

Generative adversarial networks (GANs), are a popular class of deep generative models, that are similar to the other models discussed so far in that they transform latent variables to data space using neural networks (Goodfellow et al., 2014). Like flow-based models, GANs deterministically transform latent variables into the data space. However unlike flow-based models, the transformation is not typically invertible, and the latent space is not required to be the same dimensionality as the data. This means that GANs do not necessarily define a probability density with full support on the observation space. As such, they cannot be trained used maximum likelihood, and they are instead optimized with an adversarial training process. Here we describe the original GAN objective (Goodfellow et al., 2014), but note that there are many variants and interpretations of the adversarial objective, as detailed in Creswell et al. (2018). In the original GAN formulation, a 'generator' network G is to transform latent variables to random samples, and a 'discriminator' D is tasked with distinguishing between generated samples, and examples from the data distribution. The generator is adversarially trained in order to maximize the discriminator's loss, which has the effect of drawing the generator samples closer to the data examples. The discriminator and generator alternative maximize and minimize the following binary cross entropy objective:

$$\max_D \min_G V(D,G) = \mathbb{E}_{p_{data}(\mathbf{x})} \left[\log D(\mathbf{x})\right] + \mathbb{E}_{p(\mathbf{z})} \left[\log \left(1 - D(G(\mathbf{z}))\right)\right]. \tag{2.65}$$

GANs have demonstrated very impressive samples, particularly in the image domain (Brock et al., 2019; Karras et al., 2019), however the adversarial training

process can be unstable. Goodfellow et al. (2014) describing a failure mode of 'generator-collapse', where the generator maps a wide range of latent variables to the same, or similar samples. Nonetheless, if our main requirement is producing high-quality samples, 3, then GANs are worth considering as a promising alternative method to the other deep latent variable models discussed in this chapter.

2.4.3 Autoregressive models

The generative models discussed so far *implicitly* capture dependencies between variables through a transformation of latent variables. An alternative to latent variables is to *explicitly* model the dependence of certain variables conditioned on others using an autoregressive decomposition

$$p_\theta(\mathbf{x}) = \prod_i p_\theta(x_i|\mathbf{x}_{1:i-1}). \tag{2.66}$$

This transforms the problem of estimating a high-dimensional distribution into a series of one-dimensional regression problems, for which it is straightforward to choose a parametric family, such as a Gaussian distribution for continuous data, or a categorical distribution for discrete data.

As with deep latent variable models, neural network components can be used in autoregressive models to capture complex relationships between one variable and the others. For instance if we choose a Gaussian as our conditional distribution, then we can model the i'th conditional as

$$p_\theta(x_i|\mathbf{x}_{1:i-1}) = \mathcal{N}(x_i|\boldsymbol{\mu}(\mathbf{x}_{1:i-1}), \boldsymbol{\sigma}(\mathbf{x}_{1:i-1})), \tag{2.67}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are neural networks that take the previous dimensions $\mathbf{x}_{1:i-1}$ as input. A naive approach would use a separate neural network for each of the i conditionals. However the number of parameters would quickly become impractical for high-dimensional data. Moreover, we might expect that features that are predictive of one conditional might also predict another, and that weights should be shared across the networks for each conditional. A substantial portion of research into autoregressive models has therefore aimed to develop sequential models that are more efficient computationally and in their use of parameters. For instance, recurrent neural networks process sequences of inputs, sharing

network weights across elements of the sequence, while maintaining a hidden state that is updated in order to aggregate information from previous observations (Sundermeyer et al., 2012). Another family of models masks certain connections in feedforward networks in order to preserve the autoregressive property (Uria et al., 2014; Germain et al., 2015). Domain specific variants of these models for images (van den Oord et al., 2016c,b) and audio (van den Oord et al., 2016a) have proven to be highly effective.

2.4.4 PixelCNN

The PixelCNN (van den Oord et al., 2016c) is an autoregressive neural density model for images that uses a convolutional neural network to parameterize conditional distributions for each sub-pixel in an image. Pixel values are sampled one at a time: from left to right and from top to bottom. Causality in the conditional distributions is maintained using masked convolutions that only allow connections from previously observed pixels. The PixelCNN and its variants (van den Oord et al., 2016c,b; Salimans et al., 2017) are powerful models of images, and currently are the state of the art with respect to log-likelihood scores on natural images. In our experiments we make use of the PixelCNN++ (Salimans et al., 2017), which incorporates a number of changes to the original model including the use of an alternative mixture-based pixel likelihood, downsampling to increase receptive field sizes and short-cut connections. Conditioning information is incorporated by regressing a context vector to biases which are added to intermediate feature maps. For full details see Salimans et al., (2017) and the implementation at `https://github.com/openai/pixel-cnn`.

2.4.5 Autoregressive decoders

Such models can be combined with latent-variable models as e.g. the decoder.

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}), \tag{2.68}$$

$$p(\mathbf{x}|\mathbf{z}) = \prod_i p(x_i|\mathbf{x}_{1:i-1}, \mathbf{z}). \tag{2.69}$$

This relaxes the requirement for the latent variables to explain all of the dependencies between the visible variables. Variational lossy autoencoders (Chen et al.,

2017), use autoregressive decoders with limited receptive fields in order to control the representations encoded in the latent variables. We will explore autoregressive decoders as a way to analyze representations learned by supervised image models in Chapter 6.

Chapter 3

The Shape Variational Autoencoder

A Deep Generative Model of Part-Segmented 3D Objects

This chapter is adapted from the paper *"The Shape Variational Autoencoder: A Deep Generative Model of Part-Segmented 3D Objects"* (Nash and Williams, 2017), published at the Symposium of Geometry Processing.

We introduce the shape variational auto-encoder (ShapeVAE), a generative model of part-segmented 3D objects . The ShapeVAE describes a joint distribution over the existence of object parts, the locations of a dense set of surface points, and over surface normals associated with these points. It can be used to generate novel object instances, as well as to impute missing parts or surface normals. We present an overview of our modelling and data pre-processing approach (Section 3.3). We then describe the ShapeVAE generative model in detail (Section 3.4), including the encoder (Section 3.4.5) and decoder (Section 3.4.4) networks, training process (Section 3.4.6) and baseline models (Section 3.4.7). We then provide a quantitative evaluation of the ShapeVAE on shape-completion (Section 3.5.2) and test-set log-likelihood (Section 3.5.3) tasks as well as qualitative results for object generation, shape completion, and mesh surface reconstruction (Section 3.5).

3.1 Introduction

The computer graphics industry relies to a large extent on the 3D content created by artists, modellers, designers and animators. The content creation process is time consuming and intensive even for highly skilled graphics artists. 3D Content is often created from scratch, despite the fact that vast collections of 3D models exist in online repositories and private collections. These shape collections contain considerable information about the styles, structures and textures of object classes. It is desirable to leverage this information with tools that can aid designers in the modeling process. Such tools can help by enforcing data-driven constraints, providing completions of partially designed objects, or even through the synthesis of entire shapes.

The ability to automatically synthesize and analyze 3D objects is useful not only for graphics applications, but in computer vision, where there has been a recent focus on the use of 3D shape representations in scene understanding tasks (Zia et al., 2013; Su et al., 2014; Choy et al., 2015). A detailed representation of object shape allows for complex 3D reasoning, and a model of shape variability aids the performance of recognition tasks in images. Structures such as 3D bounding boxes (Payet and Todorovic, 2011; Liebelt and Schmid, 2010), wireframe models (Zia et al., 2013), or 3D CAD models (Choy et al., 2015) have been used as shape representations and successfully recognized in images.

In this work we present the shape variational auto-encoder, a model of structural and local shape variability that captures a distribution over the co-existence of object parts, the locations of 3D surface points, and the surface normals associated with these points. We make use of the representation described by Huang et al. (2015) consisting of a collection of dense point correspondences, segmented into the object's constituent parts and augment it with point normals. We take a powerful class of deep generative model, the variational autoencoder, and introduce a novel architecture that leverages the hierarchical part-structure of 3D objects. Our model is capable of generating plausible, novel point cloud objects, and by generating consistent point normals, we can take advantage of powerful surface reconstruction methods to reconstruct smooth mesh geometry. We demonstrate that the ShapeVAE achieves strong performance in a shape completion task in comparison to a linear baseline, while producing samples of a higher quality. We

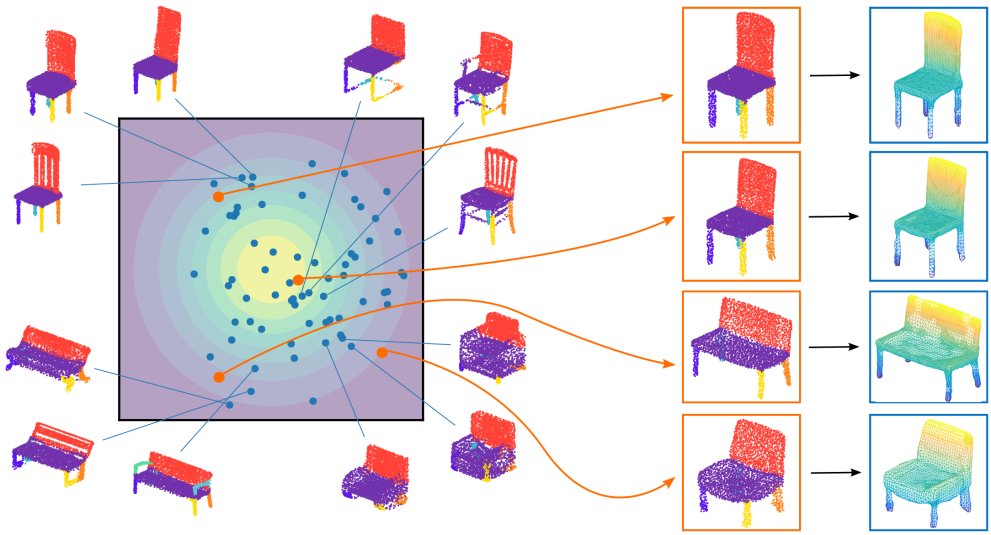


Figure 3.1: Data driven synthesis of 3D objects. (Left) Given an input collection of oriented surface points from an object class we learn a low-dimensional shape embedding using a deep probabilistic auto-encoder. (Middle) Samples from the prior embedding distribution can be passed through the decoder to obtain novel shape examples complete with point orientations (not shown). (Right) Smooth meshes are constructed by making use of the sampled point orientations.

further show that the ShapeVAE learns a semantically meaningful latent space, and that by sampling both point sets and surface normals, the ShapeVAE enables the use of powerful surface reconstruction techniques. In addition the ShapeVAE is considerably more efficient to train compared to related work, and it allows for efficient sampling and missing data imputation.

3.2 Related work

Our methods relate to statistical models of objects and shapes as well as general deep generative models. We provide an overview of the most relevant prior work.

3.2.1 Shape synthesis and generative shape models

Generative models of 3D objects have been proposed for a range of shape representations including 3D voxel images (Wu et al., 2015; Girdhar et al., 2016), keypoints (Huang et al., 2015), and meshes (Kalogerakis et al., 2012; Zuffi and

Black, 2015; Yümer et al., 2015). Early work includes active shape models (Cootes et al., 1995); statistical models of corresponding landmark points that have been used to model face shape (Lanitis et al., 1997), medical images (Heimann and Meinzer, 2009), and cars (Zia et al., 2013). Such models are often applied to relatively few landmark points, and as such are more useful for analysis rather than synthesis of whole objects.

Much of the recent work on generative shape models has focused on voxel representations of 3D objects. Wu et al. (2015) model the joint distribution of voxels and object class labels with a convolutional deep belief network. The authors use the model to recognise object classes and reconstruct 3D objects from a single depth image. Girdhar et al. (2016) use a 3D convolutional auto-encoder to establish a compressed vector representation of 3D objects that can be predicted and reconstructed in real images. A 3D generative adversarial network with convolutional structure was used by Wu et al. (2016) to synthesize voxel objects. Volumetric representations have the advantage that different objects are directly comparable on the voxel level, whereas triangulated meshes do not have an explicit parameterization that is consistent across instances. However naive volumetric methods are limited in terms of resolution, as the dimensionality is cubic in the width of the voxel grid. In addition, voxel grids require modelling of many redundant dimensions, such as empty space inside or outside the object itself, although recent work using octree representations seeks to address this (Tatarchenko et al., 2017).

Our methods model object surfaces in addition to the structural variability associated with the presence or absence of certain parts. Other work has similarly made use of the part-decomposability of objects in their shape models, either by explicitly recombining part instances from a database (Kalogerakis et al., 2012; Averkiou et al., 2014; Zheng et al., 2013), or by incorporating parts as a modelling structure (Fish et al., 2014; Zuffi and Black, 2015). Kalogerakis et al. (2012) learn a generative model over continuous and discrete geometric features of object parts and synthesize shapes by matching the generated features to object parts in database. Averkiou et al. (2014) fit templates consisting of deformable cuboids to large collections of 3D objects and obtain a low-dimensional hierarchical embedding, that captures semantic similarity between the input objects. The authors designed a method for interactive object synthesis in which a user can

explore the embedding space, and create new objects by deforming parts from nearby objects. Fish et al. (2014) also used a parts-based method in which they modelled the geometric relationships between shape parts. For each shape, unary relations such as the relative length of a part, and binary relations, e.g. the angle between two parts were captured. In related work Zuffi and Black (2015) develop a parts-based 'stitched puppet' model of human shape which allows for the shape and pose of body parts to be modelled separately, while encouraging connecting parts to be close together. This allows for shape variation to be captured on various levels: on the global level articulated pose is modelled, and on the local level continuous shape deformation is modelled. Our methods are different in that although we make use of object parts, we also use a highly detailed shape representation consisting of dense point clouds.

Recent work using dense point clouds includes PointNet (Qi et al., 2017), in which unordered point sets are processed using deep networks with a symmetric pooling function. Such networks were shown to be effective in semantic segmentation and object classification tasks. However, PointNet is not a generative model of point sets, but rather it maps input point sets to output such as a model classification, or part segmentation. In related work, a conditional generative model of unordered point sets was introduced in (Fan et al., 2017), where given an image, a collection of 3D output points was synthesized that captures the coarse shape of objects in the image. The closest work to ours is by Huang et al. (2015) in which part-segmented 3D keypoints are modelled with the beta shape machine (BSM), a variant of a multi-layer Boltzmann machine that captures global and local shape variation with a similar part-oriented structure. This model is demonstrated to be effective at generating plausible shapes, as well as for shape segmentation and fine grained classification tasks. Unlike the BSM which is an undirected probabilistic model, the models in this paper are directed, and as such training and sampling is more rapid, and we may more easily scale to high-dimensional data. When we model surface normals as well as points we double the dimensionality of the data-space, and being able to efficiently train in very high-dimensional space becomes important.

3.2.2 Deep generative models

Our generative model of oriented point clouds is a variant of a variational autoencoder, which is a method for performing learning and inference in a deep generative model (DGM): a class of generative model that employs deep neural network architectures.

Other DGMs include deep Boltzmann machines (DBMs) (Salakhutdinov and Hinton, 2009). DBMs are undirected models that have been used to capture complex distributions over speech data (Mohamed et al., 2012), images (Eslami et al., 2014; Roux et al., 2011) and part-segmented objects (Huang et al., 2015). Although DBMs are flexible, they can be difficult and time-consuming to train, and are more complicated to sample from than directed models. Generative adversarial networks (GANs) are a powerful class of DGN in which a generator network maps low-dimensional latent samples to the data space, and a discriminator network is trained to distinguish between real and fake samples (Goodfellow et al., 2014). By training the generator neural network to fool the discriminator, samples are pushed closer to the true data distribution. GANs have been used to model images (Goodfellow et al., 2014; Denton et al., 2015) and voxels (Wu et al., 2016), and are notable for producing sharp, high-quality samples. However, GANs do not explicitly describe a probability distribution over data and as such can be difficult to evaluate.

In this work we take the VAE, a powerful class of deep generative model that enables efficient sampling, density estimation and conditional inference, and introduce structure that captures the hierarchical part-structure of 3D objects.

3.3 Overview

Our goal is to take a collection of segmented input objects with dense point correspondences and to learn a generative model of 3D shape such that we can synthesize novel examples, complete partial objects, and embed 3D objects in a low-dimensional latent space. In this section we provide an overview of our deep generative shape model, a description of the data sets used, and the required pre-processing.

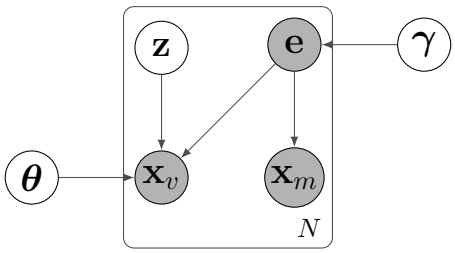


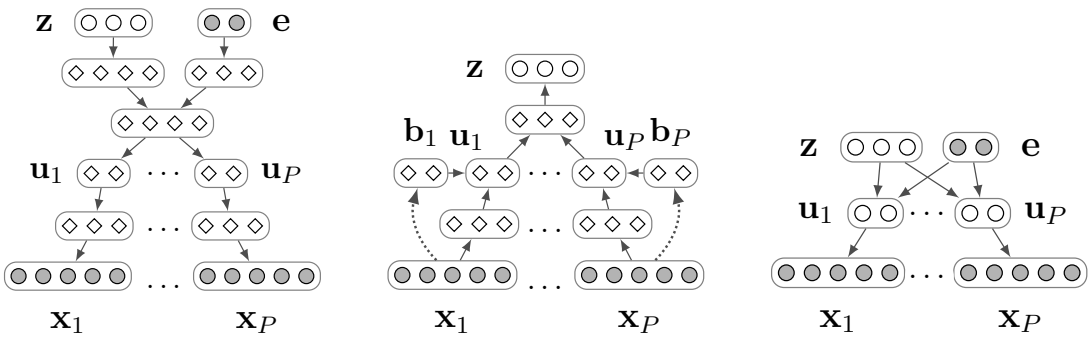
Figure 3.2: ShapeVAE graphical model. Filled circles represent visible variables, unfilled circles represent latent variables, and diamonds represent deterministic variables. Latent variables **z** are sampled from a prior distribution, and existence variables **e** are sampled from a learned distribution. Conditioned on these variables the continuous variables **x** (Surface points and normals) are generated.

3.3.1 Generative model

The core of our method is a generative model that describes a probability distribution over surface points, surface normals, and part existences for large collections of 3D objects. The relationships between these variables in 3D objects is highly complex due to hierarchical part relationships, symmetry relationships, as well as local smoothness and other structural constraints. Our model is a variant of a variational auto-encoder: a powerful generative model capable of capturing complex distributions over high-dimensional data. The VAE consists of a decoder that maps a latent code to a distribution over the data space, and an encoder that maps data to an approximate posterior distribution over latent codes. We equip our VAE with a hierarchical architecture in which higher layers capture global, structural relationships in objects, and lower levels capture variability within object parts. After training the ShapeVAE we gain the ability to sample new instances, perform shape completion and a lower bound on the likelihood of unseen instances. We also obtain a compact shape descriptor in the form of the highest level latent variables, and an encoder network that can map a data instance to this high-level description efficiently.

3.3.2 Data and pre-processing

Our methods assume a collection of consistently aligned and scaled 3D shapes for which point-wise correspondences, consistently-oriented surface normals and consistent segmentations are available. We assume that each object class has



(a) ShapeVAE decoder (b) ShapeVAE Encoder (c) ShapeFA

Figure 3.3: Generative models of part-segmented shapes. Filled circles represent visible variables, unfilled circles represent latent variables, and diamonds represent deterministic variables. (a) Latent variables and existences are mapped deterministically to intermediate part representation variables $\mathbf{u}_p$ for each part p. The part representations are then mapped separately to generate data variables. (b) Visible continuous variables are mapped to latent variables through intermediate part representation variables. Learned biases $\mathbf{b}_p$ are introduced for parts p that are not present. (c) The ShapeFA baseline model has linear connections between latent variables, stochastic part representations and continuous data variables.

a fixed number of parts, and that these parts can be present or absent for any particular object example. For example an airplane can have up to 6 parts: the fuselage, two wings, two horizontal tail fins, and one vertical tail fin. In most instances a plane will have all of these parts, but in some cases the vertical of horizontal tail fins may not be present. Figure 3.6a shows some example planes with part segmentations. We make use of such collections provided by Huang et al. (2015), and note that there exist effective methods for automatic analysis of 3D object databases that obtain correspondences and segmentations as an output (Huang et al., 2015; Kim et al., 2013). The datasets we use feature chair and airplane object classes with 3701, and 1509 examples respectively. The 3D meshes were originally collected from the Trimble Warehouse online repository by Kim et al. (2013).

3.4 Shape variational autoencoder

In this section we describe our generative model of 3D objects, the shape variational autoencoder. The ShapeVAE aims to model the joint distribution over part existences, surface points and surface normals. This task is made challenging by a number of factors. The dimensionality of the data can be extremely high, with an object class with 5000 surface points having 15000 variables describing point locations, and a further 15000 variables describing surface normals. This poses difficulties for modelling, as even with thousands of data examples, only a tiny portion of this 30000-dimensional space can be covered. Beyond this, there are a range of complex dependencies that must be captured in order to be able to synthesize plausible shapes. The model must capture symmetry, functional relationships, local continuity and smoothness, and be able to handle multi-modality in the data. Consider for example the chair object class: plausible shapes feature chair legs that match in style, and shapes, and attach to the seat appropriately. Chairs can vary at a part level in terms of the style of the chair back or legs, but also at the object level, in which objects can belong to one of a number of distinct styles. These styles induce multi-modality in the data-distribution, indeed Huang et al. (2015) demonstrated that on this dataset the marginal distributions of surface point locations can have complex multi-modal distributions.

Although the modelling task is challenging, there are a number of simplifying features that we can exploit. The data-dimensionality is very high, however the data is highly structured, such that the intrinsic dimensionality of the data is much smaller. Take a chair leg for example, in our data this object part consists of around 100 keypoints, but the locations of these keypoints are highly dependent, such that knowledge of only a few points would enable reasonable estimation of the rest. Moreover, the variability across different examples is restricted: legs tend to vary in their length, width, angle, and the location at which they join the base of the chair. As such chair legs possess far fewer degrees of freedom than the 600 dimensions which describe the keypoints and surface normals. We can also make use of the global structure of 3D objects; that they have parts, that the parts are organized in typical arrangements, and so we can design our model in a way that makes use of these known structures. By incorporating a hierarchical

part structure in our model, we introduce an inductive bias that favours part decomposability, and a factored relationship between global and local variability. We can also take advantage of recently developed deep learning methods, that can compress the data more effectively than comparable linear methods.

3.4.1 Data description

We build on the work of Huang et al. (2015) and represent objects in the following way:

- **Part existences:** For an object class with P possible parts we describe the existence or non-existence of object parts using a binary vector $\mathbf{e} \in \{0, 1\}^P$, where part p is present if $e_p = 1$ and not present if $e_p = 0$.
- **Surface points:** The surface of an object can be represented with a collection of points on the object surface. These surface points are organized in a consistent order across examples in a dataset, such that the point dimension k_i^r on object r is approximately in correspondence with point dimension k_i^s on object s. We use the approximate point correspondences obtained by Huang et al. (2015), who use a method that alternates between estimating part-level shape templates, and estimating point correspondences within a part. For some objects not all parts will be present, and so the associated surface points will also not be present. We denote these with a missing data symbol m. For an object with D possible surface points we have a vector $\mathbf{k} \in (\mathbb{R} \cup m)^{3D}$ of point positions.
- **Surface normals:** We can describe the surface of an object in more detail by also including the orientation of the surface points. As with the surface points we describe the surface normals with a vector $\mathbf{n} \in (\mathbb{R} \cup m)^{3D}$.

As our modelling process is identical for keypoints and normals, for convenience we use $\mathbf{x} = [\mathbf{k}, \mathbf{n}]$ to refer to the collection of all continuous variables. Our generative model aims to model the joint distribution of part existences, surface points, and surface normals $p(\mathbf{e}, \mathbf{x})$. One potential issue with this object representation is the extent to which we can define pointwise correspondences and part-specifications that are consistent and capture the variability present in an object class. In general this depends on the object class, but we find it to be a reasonable approximation

for the chair, plane, and bike object-classes we study here.

3.4.2 Model structure

We model the joint distribution of existences, points, and surface normals by first modelling the marginal distribution of the existences $p(\mathbf{e})$, and then by modelling the continuous variables conditional on the existences $p(\mathbf{x}|\mathbf{e})$. Let $p(i)$ be the part index associated with keypoint i, and let $\mathcal{M}(\mathbf{e}) = \{i|e_{p(i)} = 0\}$ be the set of indices of missing variables for a particular object. We can then write the set of missing keypoints and normals as $\mathbf{x}_m = \{\mathbf{x}_i\}_{i\in\mathcal{M}}$ and the set of visible keypoints as $\mathbf{x}_v = \{\mathbf{x}_i\}_{i\notin\mathcal{M}}$ where we drop the dependence on $\mathbf{e}$ for notational convenience. The missing keypoints and normals $\mathbf{x}_m$ are completely determined by the part existences, and so they are simply assigned the missing data symbol m. Visible keypoints and normals $\mathbf{x}_v$ are generated using a latent variable model. This latent variable model first draws samples $\mathbf{z}$ from a prior Gaussian distribution over a latent space. These prior samples are then mapped to the parameters of a diagonal Gaussian distribution using parameterised functions $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{e})$ and $\boldsymbol{\sigma}^{\mathbf{2}}_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{e})$ where these parameter functions are given by the decoder of the ShapeVAE:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}), \tag{3.1}$$

$$p(\mathbf{x}_v|\mathbf{e}, \mathbf{z}; \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_v|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{e}), \boldsymbol{\sigma}^{\mathbf{2}}_{\boldsymbol{\theta}}(\mathbf{z}, \mathbf{e})), \tag{3.2}$$

$$p(\mathbf{x}_m|\mathbf{e}) = \mathbb{I}[\mathbf{x}_m = [m, \ldots, m]]. \tag{3.3}$$

Figure 3.2 shows a graphical model representing the model structure. In the following sections we describe the marginal distribution of the part existences, the encoder-decoder structure of the ShapeVAE, the procedure for training the model parameters and the baseline models that we use to evaluate the performance of the ShapeVAE. The following sections assume some knowledge of standard terminology in deep learning, and familiarity with the standard VAE is useful. Full coverage is beyond the scope of this work, and so we refer to Doersch (2016) for a tutorial.

3.4.3 Part existences

Part existence variables $\mathbf{e}$ are assumed to have been generated by a categorical distribution $p(\mathbf{e}|\boldsymbol{\gamma})$. For an object with P parts this is a distribution over 2^P

possible states, which in the absence of independence assumptions requires $2^P - 1$ parameters. For a particular arrangement of part existences $\mathbf{e}_p$, the maximum likelihood estimate is simply the ration of the number of times the arrangement occurs in the training set $N_{\mathbf{e}_p}$ divided by the number of examples in the training set N:

$$p(\mathbf{e} = \mathbf{e}_p; \boldsymbol{\gamma}) = \gamma_p, \quad \gamma_p = \frac{N_{\mathbf{e}_p}}{N}. \tag{3.4}$$

In practice only a few arrangements of part existences occur in object datasets, and so a maximum likelihood estimate assigns most part combinations zero probability. The main limitation of this model is that part arrangements that do not appear in the training set will never be sampled by the generative model. However it is straightforward to choose a prior distribution over the distribution parameters such as a Dirichlet or to even manually choose a desired distribution over part arrangements such that unseen arrangements can be sampled.

3.4.4 ShapeVAE Decoder

The decoder of a variational autoencoder is responsible for mapping from latent variables to the parameters of a conditional data distribution. This mapping is implemented using a fully-connected neural network. This conditional data distribution is typically chosen such that the data variables $\mathbf{x}$ are conditionally independent given the latent variables $\mathbf{z}$, and so that it naturally models the domain of the data variables. The conditional independence of the data variables forces the latent variables to explain the interdependence of the visible data variables. This causes the model to learn a latent space in which the main modes of variability are captured. In our case we additionally condition on existence variables $\mathbf{e}$, and as the keypoints and surface normal data is continuous we map to the parameters of a diagonal Gaussian distribution $\boldsymbol{\mu}(\mathbf{z}, \mathbf{e})$ and $\boldsymbol{\sigma}^2(\mathbf{z}, \mathbf{e})$. It is useful to set a minimum variance $\boldsymbol{\sigma}^2_{\text{min}}$ for each dimension of the decoder distribution, so that the total variance is given by $\boldsymbol{\sigma}^2_{\text{tot}}(\mathbf{z}, \mathbf{e}) = \boldsymbol{\sigma}^2(\mathbf{z}, \mathbf{e}) + \boldsymbol{\sigma}^2_{\text{min}}$. This prevents the decoder from assigning very small variance to any reconstructed training example, which helps reduce overfitting. We treat the minimum variance as a hyperparameter that can be adjusted depending on the task.

We take advantage of the part-structure of the data and use a hierarchical decoder architecture in which global latent variables $\mathbf{z}$ and existence variables

$\mathbf{e}$ capture structure, style and shape characteristics of the whole object, and a lower level part-representation $\mathbf{u}_p$ captures variability within each part p. The decoder takes latent variables and existence variables as input, and passes them separately through fully-connected layers of size 256, before concatenating to form a layer of size 512. This intermediate layer is then mapped to the part representation $\mathbf{u}(\mathbf{z}, \mathbf{e})$ of size $\sum_p n_{u_p}$ where n_{u_p} is the size of part p's representation. We treat the size of each u_p as a hyperparameter, and typically use 128 or 256 units per part. This representation is then split into its constituent parts $\mathbf{u} = [\mathbf{u}_1, \ldots, \mathbf{u}_P]$ and mapped to a fully-connected pre-parameter layer $\mathbf{h}_p(\mathbf{z}, \mathbf{e})$ of size 512. Finally the output parameters are obtained using a linear layer for the mean $\boldsymbol{\mu}_p(\mathbf{z}, \mathbf{e}) = \texttt{Linear}(\mathbf{h}_p(\mathbf{z}, \mathbf{e}))$, and by applying a soft-plus non-linearity for the variance $\boldsymbol{\sigma}_p^2(\mathbf{z}, \mathbf{e}) = \texttt{softplus}(\texttt{Linear}(\mathbf{h}_p(\mathbf{z}, \mathbf{e})))$ to ensure that the variance is positive. A simplified view of the decoder structure is shown in Figure 3.3a.

3.4.5 ShapeVAE Encoder

The encoder of the ShapeVAE aims to approximate the posterior distribution over the latent variables $p(\mathbf{z}|\mathbf{x}, \mathbf{e})$ associated with the generative distribution $p(\mathbf{x}|\mathbf{z}, \mathbf{e})$ and prior $p(\mathbf{z})$. As the true posterior distribution $p(\mathbf{z}|\mathbf{x}, \mathbf{e})$ is intractable to evaluate, we make use of an approximate posterior $q(\mathbf{z}|\mathbf{x}, \mathbf{e})$, and use variational inference to learn the parameters of both the encoder and decoder simultaneously. Similar to the decoder, the ShapeVAE decoder is a neural network that maps from inputs $\mathbf{x}$ to the parameters of a diagonal Gaussian $\boldsymbol{\mu}(\mathbf{x})$ and $\boldsymbol{\sigma}^{\mathbf{2}}(\mathbf{x})$.

The ShapeVAE encoder reverses the architecture of the decoder but is modified in order to process input parts which may be missing. As shown in Figure 3.3b the encoder takes keypoints and normals $\mathbf{x}$ as input and maps to a part representation $\mathbf{u} = [\mathbf{u}_1, \ldots, \mathbf{u}_P]$. For parts that are present, the input is mapped through an intermediate fully connected layer of size 512, whereas parts that are missing simply generate a learnable bias $\mathbf{b}_p$ which is added in the appropriate position to the part representation. The part representation is then concatenated and passed through to a fully-connected pre-parameter layer of size 512. As with the decoder the pre-parameter layer is mapped through linear and soft-plus layers to obtain means and diagonal variances of the encoder distribution.

3.4.6 Training

We optimize the parameters of the ShapeVAE encoder $\boldsymbol{\theta}$ and decoder $\boldsymbol{\phi}$ using the auto-encoding variational Bayes algorithm described in Section 2.3.2. We optimize a variant on the lower bound that additionally conditions on part existences. In addition we include a parameter α that weights the KL-divergence term. For a single data example $(\mathbf{x}, \mathbf{e})$ the lower bound is:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}, \mathbf{e}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}, \mathbf{e})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, \mathbf{e}) \right] - \alpha D_{KL}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}, \mathbf{e}) \| p(\mathbf{z})), \tag{3.5}$$

which is a lower bound for $\alpha \geq 1$. The first term in the lower bound is the expected reconstruction probability with respect to the encoder distribution. This term encourages the decoder to reconstruct its input from samples drawn from the encoder distribution. The second term is the KL divergence between the ShapeVAE encoder distribution, and the prior distribution over the latent space $p(\mathbf{z})$. The KL divergence is an asymmetric measure of distance between two probability distributions, and maximizing the negative KL divergence encourages the latent posterior to be close to the prior. For a standard Gaussian prior distribution, this has the effect of pushing the means of the encoder distribution towards 0, and the variances towards 1.

Modifications of the KL divergence weighting α term have been used to reduce posterior collapse, in which certain latent variables collapse to the prior distribution, and stop encoding information about the inputs (Sønderby et al., 2016). It has also been used to shape the learned latent representations, with experimental evidence suggesting that larger values of α can lead to a disentanglement of data variability across latent dimensions (Higgins et al., 2017). In this work we use larger values of α to increase the pressure on the encoder distribution to be close to the prior, and as a result to help ensure that samples from the prior occupy the same space as the encoded data values. We set α heuristically in order to improve sample quality, but optimizing α dynamically during training in order to satisfy a reconstruction constraint as in Rezende and Viola (2018) may allow more precise control in future work.

3.4.7 Baseline models

In order to evaluate the ShapeVAE we introduce two baseline models. The first is a very simple baseline, in which a separate diagonal Gaussian model is fitted to the points and surface normals for each combination of existences in the training set. The model is given by:

$$p(\mathbf{x}_v|\mathbf{e},\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_v|\boldsymbol{\mu}_\mathbf{e}, \boldsymbol{\sigma}^2_\mathbf{e}). \tag{3.6}$$

Thus for each pattern of existences in the training set we take the mean of the surface points and normals, and compute the variance of each dimension.

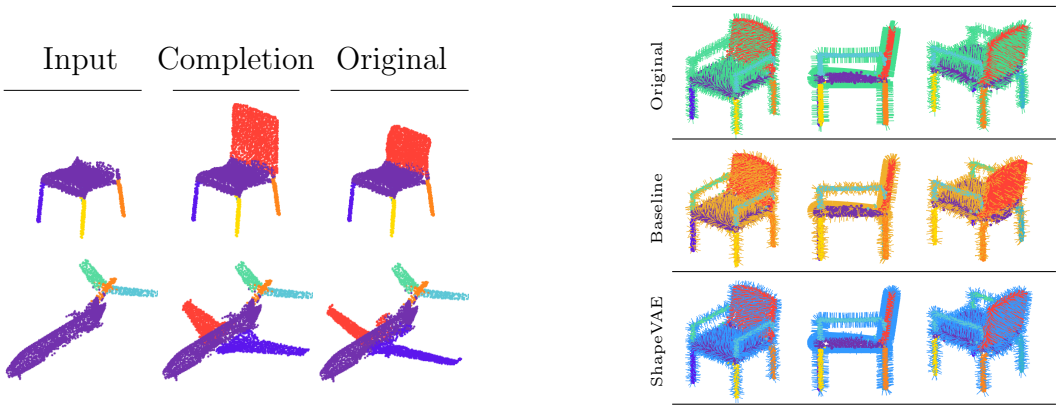
We also introduce a more competitive baseline, the shape factor analyzer (ShapeFA). In this model we replicate the hierarchical structure of the ShapeVAE, but use linear connections between layers, rather than arbitrary non-linear functions. This is inspired by the widely-used factor analysis model in which latent variables are linearly mapped to data variables, and diagonal Gaussian noise is added. In our version, the model has two layers of linear mappings: from top-level structural latent variables to a part representation, and from the part representation to the object parts. As in the ShapeVAE the top-level latent variables capture overall shape variability in terms of style, symmetry and functional dependencies, while the lower level latent variables capture local variability within a particular part. Writing $\mathbf{u}_v$ for the set of visible part representation variables we have the following model:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0},\mathbf{I}), \tag{3.7}$$

$$p(\mathbf{u}_v|\mathbf{z},\mathbf{e},\boldsymbol{\theta}) = \mathcal{N}(\mathbf{u}_v|\mathbf{W}^{(\mathrm{T})}_\mathbf{e}\mathbf{z} + \boldsymbol{\mu}^{(\mathrm{T})}_\mathbf{e}, \boldsymbol{\Psi}^{(\mathrm{T})}_\mathbf{e}), \tag{3.8}$$

$$p(\mathbf{x}_p,\mathbf{n}_p|\mathbf{u}_p,\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}_p|\mathbf{W}^{(\mathrm{B})}_p\mathbf{u}_p + \boldsymbol{\mu}^{(\mathrm{B})}_p, \boldsymbol{\Psi}^{(\mathrm{B})}_p), \tag{3.9}$$

where we use $\boldsymbol{\theta}^{(\mathrm{B})}$ and $\boldsymbol{\theta}^{(\mathrm{T})}$ to denote bottom and top-layer parameters respectively. The advantage of this model is that by integrating out the part representation variables we can evaluate the log-likelihood exactly. We can also train it rapidly using the greedy layer-wise procedure described by Tang et al. (2012). Figure 3.3c shows the structure of the ShapeFA.



(a) Shape completions

(b) Surface normal estimation.

Figure 3.4: Shape and surface-normal completions. (a) Missing parts completed by performing conditional inference with the ShapeVAE model. (b) Surface normals predicted by conditional inference with the ShapeVAE, compared to normals estimated using local plane fitting (Hoppe et al., 1992). For the latter approach, the relative sparsity of the surface points leads to noisy estimates.

	Shape completion		
Model	Chairs	Planes	Bikes
ShapeVAE-8	0.124	0.106	**0.168**
ShapeVAE-64	0.121	**0.104**	**0.168**
ShapeFA-8	**0.116**	0.243	0.178
ShapeFA-64	0.155	0.267	0.255

Table 3.1: Shape completion average square error per dimension, for ShapeFAs and ShapeVAEs with 8 and 64 latent dimensions.

	Log-likelihood		
Model	Chairs	Planes	Bikes
ShapeVAE-8*	0.454	0.466	0.158
ShapeVAE-64*	0.486	0.551	0.206
ShapeFA-8	**0.514**	**0.739**	**0.475**
ShapeFA-64	0.466	0.648	0.300
Diag. Gaussian	0.001	0.057	−0.015

Table 3.2: Test set log-likelihood in nats per dimension for an independent Gaussian baseline, the ShapeFA, and the ShapeVAE. ShapeVAE scores are lower bounds estimated using 10,000 importance weighted examples.

3.5 Evaluation

We detail the performance of the ShapeVAE on shape completion, and test set log-likelihood tasks, as well demonstrate the model's ability to synthesize 3D shapes. We examine features of the latent-space learned by the ShapeVAE and compare surface reconstruction methods that can be used to convert sampled point clouds to meshes.

3.5.1 Training details

We trained ShapeVAEs with 8 and 64 latent dimensions and part representations of size 256 per part using the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 10^{-4} for up to 500 passes through the training set in all our experiments. For the quantitative experiments we used early stopping, where model training was halted based on performance a validation set. For sample synthesis we allowed the model to train for the full 500 passes through the training set. For the airplane and chair datasets we used a batch size of 100, and for the bike dataset we use a batch size of 64. We use a minimum variance of $\boldsymbol{\sigma}^{\mathbf{2}}_{\text{min}} = 10^{-3}$ and KL weighting $\alpha = 10^2$ in all our models. We note that value of α is high relative to other work, however we found it to be important to use a large value in order to obtain good quality samples. ShapeFAs with 8 and 64 latent dimensions, and part representations of size 128 per part were trained as a baseline model.

The ShapeVAE takes around 30 minutes to train on an Nvidia Titan X GPU for our largest dataset consisting of 3701 chairs. This is a order of magnitude faster than the beta shape machine introduced by Huang et al. (2015) which is reported to take 45 hours for the same dataset.

3.5.2 Shape completion

We evaluate the ShapeVAE's ability to complete shapes, where the task is to infer missing keypoints or surface normal variables given observations of the other variables. For all plots of object samples or reconstructions we show the mean $\mathbb{E}_{p(\mathbf{x}|\mathbf{z})}[\mathbf{x}]$ of the data variables given the latents. We observed that samples from the conditional data distribution were quite noisy, which indicates that a substantial part of the data variability is explained by the noise distribution. We attribute this to the imperfect correspondences present in the data, which make it challenging to explain the data variability with low-dimensional latents. In combination with the relatively large α term, which disincentivises the model from encoding too much information in the latents, this leads to a large portion of data variability being explained at the local noise level. It would be of interest to investigate these trade-offs in future work, as well the applicability of more flexible posteriors, priors or conditional data distributions (Kingma et al., 2016) in order to improve modelling performance.

For the ShapeVAE we perform conditional inference by initializing the missing values with noise, and iteratively sampling the latent variables conditioned on the data, and then the data variables given the latents. As described above we use the conditional mean for samples at the data level, but take true samples from the posterior. This defines an Markov chain Monte Carlo (MCMC) procedure that samples from the conditional distribution as required (Rezende et al., 2014). Two particularly useful tasks are the completion of missing parts, and estimation of surface normals for a given point cloud. Figure 3.4a shows examples of plausible completions for the part-completion task, and Figure 3.4b shows normals recovered by the ShapeVAE given an input point cloud.

In order to quantitatively test the shape completion abilities of the ShapeVAE, we use the following experiment. We sample with replacement 1000 objects from the test sets of each object class. We then drop-out parts independently with

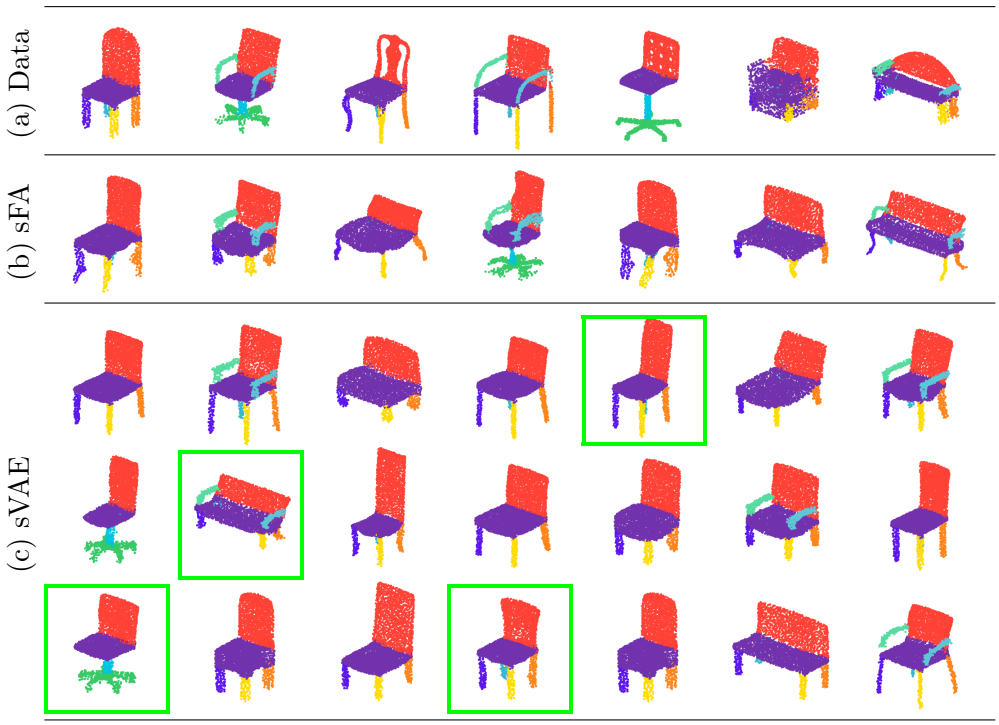


Figure 3.5: (a) A collection of examples from the chairs dataset. (b) Samples generated by a ShapeFA with 64 latent dimensions. (c) Samples generated by a ShapeVAE with 64 latent dimensions. Stylistic variability highlighted in green.

probability 0.25, and reject a part selection if all or none of the parts remain. The ShapeVAE and ShapeFA are used to impute the missing parts and we compute the mean squared error between the reconstructed values and the true values. For both the ShapeVAE and ShapeFA we sample from the conditional distribution of the missing parts given the visible parts, and estimate the conditional mean by averaging over 25 samples, with an MCMC burn-in of 100 samples, and a gap of 10 between each chosen sample. The ShapeVAE outperforms the ShapeFA in both the planes and bikes categories, and achieves similar results in the chairs category as shown in Table 3.1.

3.5.3 Likelihood

A standard measure of a generative model's performance is test set log-likelihood: the average probability of a set of unseen datapoints under the model. We evaluate the log-likelihood obtained by the ShapeVAE against baseline models on each of the four object classes. Although the ShapeVAE is a probabilistic model it

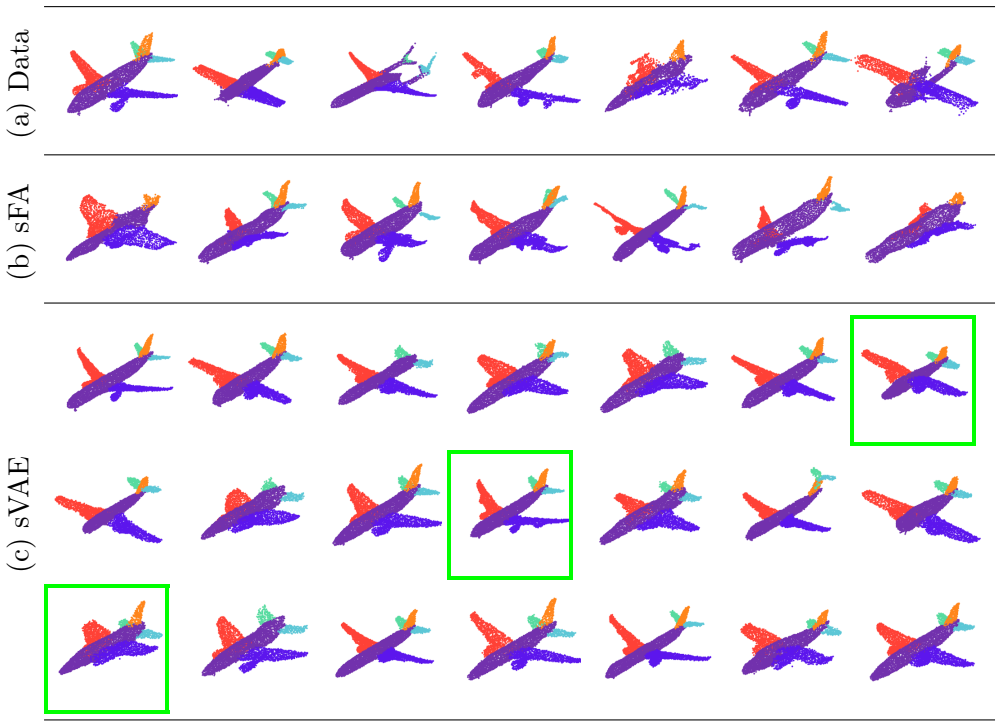


Figure 3.6: (a) A collection of examples from the airplanes dataset. (b) Samples generated by a ShapeFA with 64 latent dimensions. (c) Samples generated by a ShapeVAE with 64 latent dimensions. Stylistic variability highlighted in green.

is not possible to evaluate the exact probability of a data point, only a lower bound. As such we estimate the log-likelihood using 10,000 importance-weighted samples (Burda et al., 2016). This provides a lower bound on the log-likelihood that is tighter than the training lower bound, however it may still be significantly lower than the true value. We compare with the ShapeFA and diagonal Gaussian baselines for which we can obtain an exact log-likelihood score. Table 3.2 shows that although the ShapeVAE has much better performance than the Gaussian baseline, the ShapeFA with 8 latent dimensions achieves the best performance on all datasets. However it should be emphasized that the reported score for ShapeVAE models is a lower bound, and the true log-likelihood score may be significantly higher.

3.5.4 Sample quality

We qualitatively investigate the extent to which the ShapeVAE synthesizes quality 3D objects, as compared with real data examples and the ShapeFA. Good samples

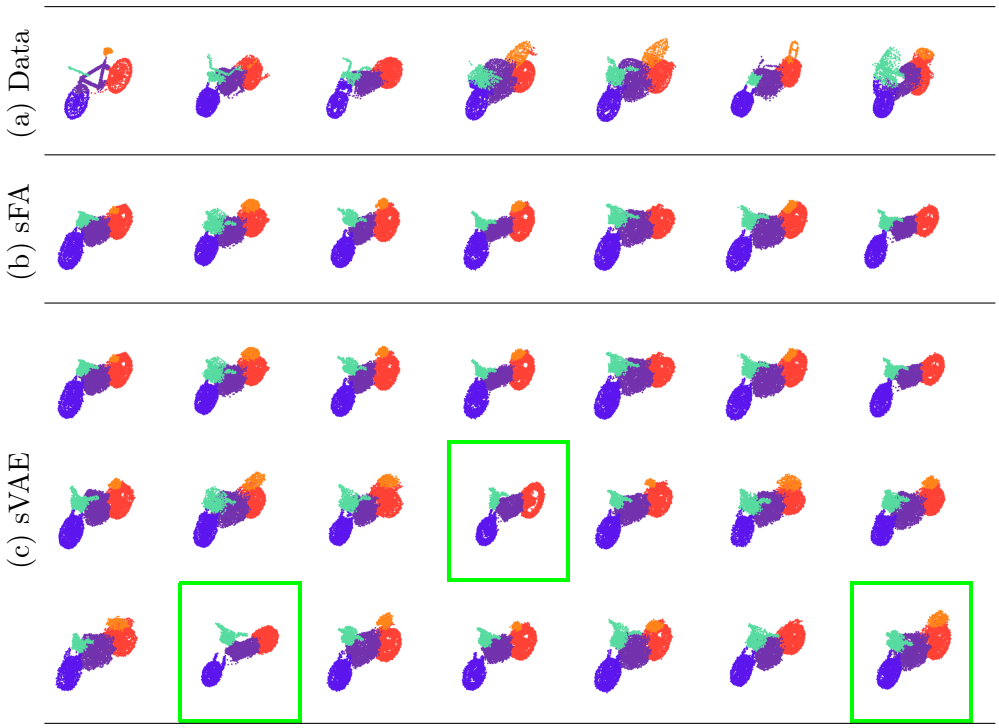


Figure 3.7: (a) A collection of examples from the bike dataset. (b) Samples generated by a ShapeFA with 64 latent dimensions. (c) Samples generated by a ShapeVAE with 64 latent dimensions. Shape variability highlighted in green.

are characterised by realism: where objects demonstrate regular surfaces, appropriate symmetry, functional plausibility in terms of part attachments, as well as fine detail. Samples should also demonstrate a wide range of shape variability, and capture the main modes of variation in the object class. For example a model trained on chairs data should be able to generate both wide benches, tall, thin chairs, as well as armchairs and office chairs. Another desirable feature of a model's samples is novelty: shape samples should not simply recreate instances from the training data set.

For both the ShapeVAE and ShapeFA we sample latent variables from their prior distributions, and use the conditional data distribution mean as output. For the ShapeFA we additionally draw samples for the part latent variables. Figures 3.5, 3.6 and 3.7 show samples from the ShapeVAE alongside data examples, and samples from the ShapeFA. The ShapeFA produces samples that demonstrate a wide range of variability, however they feature irregular surfaces and occasional asymmetry. This is particularly evident for the airplane object class in which a number of examples demonstrate misshapen features. By contrast the ShapeVAE

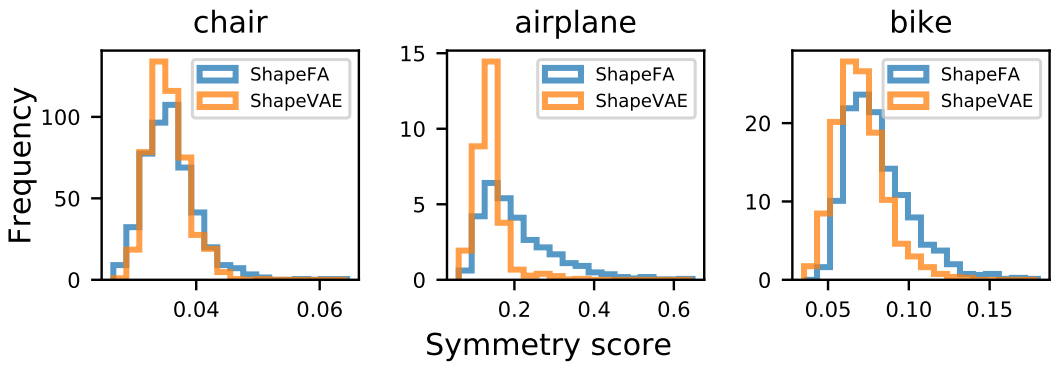


Figure 3.8: Symmetry score distribution for samples from the ShapeVAE and the ShapeFA.

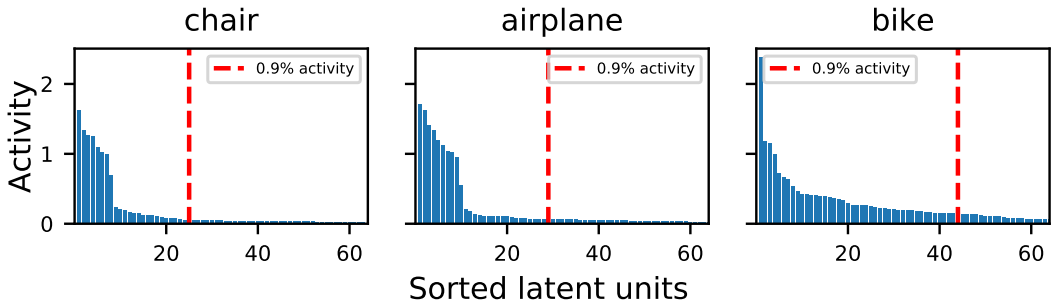


Figure 3.9: The activity of each latent unit (as defined in Section 3.5: Latent structure) sorted in descending order for ShapeVAEs with 64 latent dimensions. The first units at which the cumulative activity is greater than 90% is indicated.

produces samples that are realistic, with regular surfaces and good symmetry. The samples also demonstrate a good range of shape variability in terms of object style, with office chairs, benches, tall chairs and standard four-leg chairs all well represented for the chair object class, commercial jets, fighter jets and small planes present in the airplane samples, and bulky motorbikes and smaller off-road bikes present in the bike samples. However, neither the ShapeVAE or ShapeFA produce samples with the level of fine-detail present in the data examples. Features like chair backs with slats, or ornamental legs are not present in the model samples. This is consistent with the blurry image samples produces using VAEs in the machine learning literature (Lamb et al., 2016). It is arguable that the assumption of independence of the data variables given the latent variables enables the VAE to simply model fine details as noise.

To assess the extent to which the ShapeVAE produces symmetrical samples, we

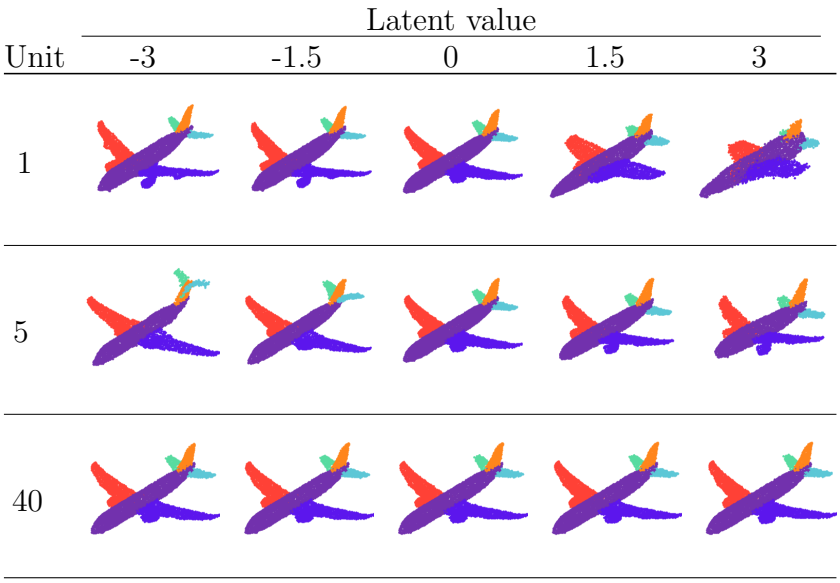


Figure 3.10: Data reconstructions obtained by varying a single latent unit through $[-3, \ldots, 3]$ while setting the others to zero. Units are numbered in order of their activity (see Figure 3.9.

compute a symmetry score. The symmetry score for a particular object is defined as the average euclidean distance between sampled points, and their nearest neighbors, after flipping on some axis of symmetry. For chairs we use the axis that splits the seat and back in half, for airplanes, we flip on the axis that extends lengthways through the fuselage. Figure 3.8 shows the distribution of symmetry scores for 1000 samples from a ShapeVAE with 64 latent dimensions, and a ShapeFA with 64 latent dimensions. The ShapeVAE achieves better symmetry scores, with a notable difference for the airplanes dataset.

Latent structure. In order to examine the latent structure learned by the ShapeVAE we train a model with 2 latent dimensions, and plot the latent space along with embeddings of a selection of the training set in Figure 3.1 (left). The training set embeddings are obtained by passing the input examples through the ShapeVAE encoder. We see that semantically similar chair styles such as benches, arm-chairs, tall chairs and ordinary chairs form clusters in latent space, which indicates that the ShapeVAE with 2 latent dimensions has learned global geometric structure such as the relative height, width and depth of the 3D objects. By selecting points close to these clusters, we can decode and obtain samples of 3D objects (Figure 3.1 (middle)). The samples share some of the main semantic characteristics of the training examples that they are embedded close to. In

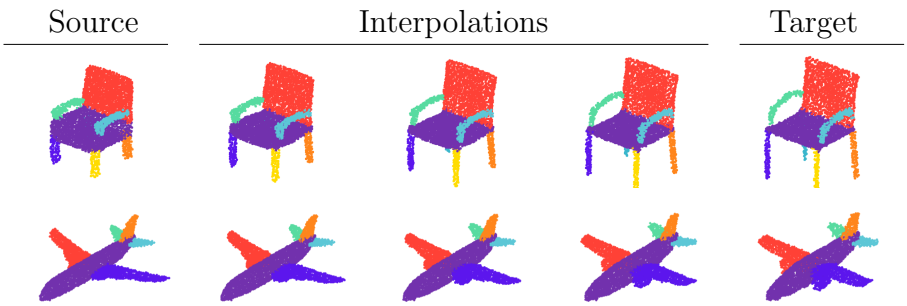
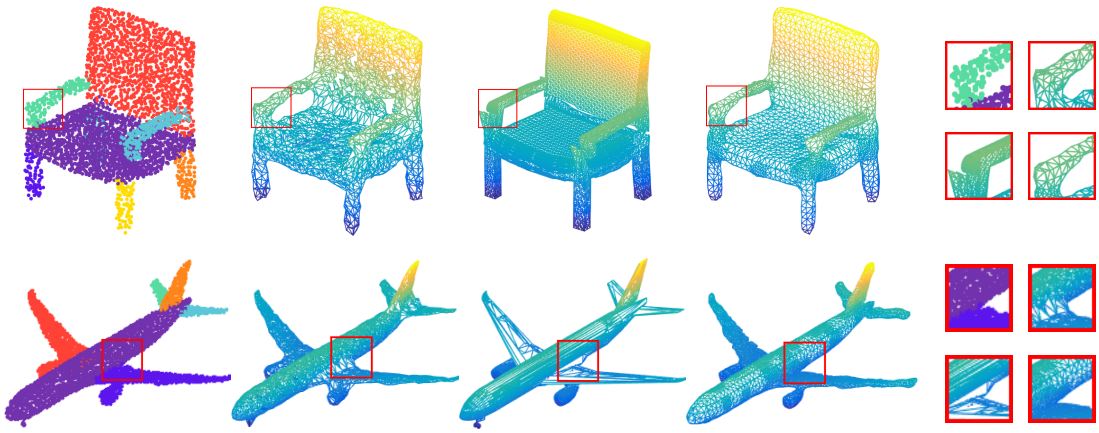


Figure 3.11: Latent space interpolation. (Source) Source 3D point cloud. (Target) Target 3D point cloud. (Interpolations) ShapeVAE Interpolations obtained by a linear interpolation in latent space.

this case the relatively narrow range of variability of the sampled shapes can be attributed partly to the very low-dimensional latent space.

For models with more latent dimensions it isn't possible to visualise the latent space in the same way, however we can demonstrate some qualities of higher dimensional latent spaces by tracing a straight line in latent space, and visualising the decoded objects sampled along this line. Figure 3.11 shows examples of these interpolations achieved using ShapeVAEs with 64 latent dimensions. The interpolations are smooth, and each intermediate point produces a plausible 3D object.

In previous work on VAEs it has been demonstrated that the models often do not encode much information in a number of the latent variables (Burda et al., 2016; Sønderby et al., 2016). We use the measure of activity described in Burda et al. (2016) which is the variance across the data set of examples transformed using the mean of the encoder distribution for a particular dimension. If a latent unit varies across different data examples, then it is reasonable to think that it is encoding information useful for reconstruction. We plot the activity of the latent units for ShapeVAEs with 64 latent dimensions in Figure 3.9. The figure shows that for each object class, the activity of the latent units drops significantly after about 10 units. This indicates that the effective dimensionality of the ShapeVAE can be lower than the pre-specified number of latent units. In Figure 3.10 we demonstrate some examples of features learned by different latent dimensions by varying a particular unit while keeping the others fixed. We see that features encoded by latent units with high activity (unit 1, 5) encode more significant



(a) Point samples (b) Alpha shapes (c) Template deformation (d) Poisson surface reconstruction (e) Detail

Figure 3.12: Mesh reconstruction. (a) A sampled 3D point cloud with surface normals. Surface reconstruction using (b) alpha shapes, (c) template deformation, and (d) Poisson surface reconstruction. (e) Surface reconstruction detail for (top left) point samples, (top right) alpha shapes, (bottom left) template deformation and (bottom right) Poisson surface reconstruction.

shape changes than those with low activity (unit 40). This reinforces the notion that the latent activity metric captures the importance of the latent units.

3.5.5 Surface reconstruction

We reconstruct 3D meshes from sampled point clouds using three methods: alpha shapes (van Kreveld et al., 2011), template deformation (Sumner et al., 2007) and Poisson surface reconstruction (Kazhdan et al., 2006). For alpha shapes we use a radius $r = 0.12$. For template deformation we use a variant of an embedded deformation in which the deformation graph's nodes are the keypoints of template part, and we smoothly deform so as to match the corresponding samples' keypoints. For Poisson surface reconstruction we the implementation of Kazhdan et al. (2006) with default parameters except for the number of samples per node, which we set to 1.5.

Exemplar mesh reconstructions using alpha shapes, template deformation and Poisson surface reconstruction for surface point clouds sampled from trained

ShapeVAEs are shown in Figure 3.12. We qualitatively evaluate the mesh reconstructions in terms of mesh quality and faithfulness of the reconstructed surface to the shape of the sampled points. The alpha shapes reconstructions (Figure 3.12b) are successful in capturing the coarse shape of the sampled points, however as the method relies on reconstruction of the input points, it produces noisy and uneven output. Triangulation-based methods also suffer at object part boundaries where surfaces intersect. This is clearly illustrated by the wing-fuselage intersection on the airplane example. The template deformation reconstructions (Figure 3.12c) make use of a database of existing object parts, which are then deformed and repositioned so as to match the sampled points. As such they inherit good-quality, human-designed mesh parts, which combine to create a high-quality, noise-free mesh reconstruction. However, template meshes can only be deformed to a limited extent before becoming irregular, and so the extent to which a template deformation can match an object sample is limited. In addition if segmentations of meshes in the database are not exactly aligned with real part boundaries, then issues can arise at the intersection of parts in the synthesized mesh. This is highlighted in the chairs example in which the chair arms are not quite compatible with the chair seat. Figure 3.12d shows Poisson surface reconstructions using the sampled point normals. The reconstructions are of better quality than alpha shapes with respect to noise and artifacts, and faithfully recover the shape of the point samples. However it should be noted that in cases where the sampled points are sparse, the Poisson surface reconstruction can fail, resulting in unusable reconstructions.

3.6 Discussion

In this paper we have demonstrated that the ShapeVAE can effectively model the high dimensional distribution over object shapes, producing realistic and novel samples. We show that modern deep learning methods are effective at scaling to very high dimensionality data, and that by modelling both 3D surface points and surface normals we enable the use of normal-based surface reconstruction methods.

The most significant limitation of our method is the reliance on input datasets containing consistent mesh segmentations as well as dense correspondences. Although there exist methods for the automatic establishment of correspondences

and segmentations, these methods are imperfect and can result in poor outputs. A significant issue is that the notion of one-to-one point correspondences for objects in diverse datasets such as chairs is ill-founded. The result is that the input data for our method can contain poor correspondences, which has a knock-on effect on sample quality. We believe that a promising avenue for future research is to represent objects using unordered point sets, which would enable the use of large datasets without pre-processing, and correspondence quality issues. Some work has already taken place in this area, with deep learning methods applied to 3D point sets for the purpose of object classification, semantic scene parsing and part segmentation (Qi et al., 2017). We believe there is potential to modify these methods for generative modelling, which would enable the synthesis of arbitrary point clouds. One concrete possibility is to treat surface points as samples from a three-dimensional distribution, that is itself characterized by latent variables.

Although the ShapeVAE's samples display a good range of variability, they are somewhat lacking in fine detail in comparison with the input point sets. This is an issue that has been documented in the machine learning literature, where VAE-based generative models of images demonstrate blurriness and a lack of detail (Dosovitskiy and Brox, 2016b), and variational methods have been shown to lead to underfitting in general (Turner et al., 2011). One promising route to improve the ShapeVAE's modelling capacity is to incorporate more expressive prior or posterior distributions as in Kingma et al. (2016). Alternatively, Flow-based, or autoregressive models (Section 2.4) may achieve good performance in this domain, but care will need to be taken to design efficient and performant architectures, given the high dimensionality of the data.

Chapter 4

Autoencoders and Probabilistic Inference with Missing Data

An Exact Solution for The Factor Analysis Case

This chapter is adapted from the article: "*Autoencoders and probabilistic inference with missing data: An exact solution for the factor analysis case*" (Williams et al., 2018). The main ideas and theoretical analysis for this chapter were contributed by Chris Williams, with input from myself and Alfredo Nazábal. I additionally contributed empirical results, as well as an analysis of these results.

Latent variable models can be used to probabilistically impute missing data entries. The variational autoencoder architecture (Kingma and Welling, 2014; Rezende et al., 2014) includes a recognition or encoder network that infers the latent variables given the data variables. However, it is not clear how to handle missing data variables for a given trained model. The factor analysis (FA) model can be thought of as a basic autoencoder, using linear encoder and decoder networks, that can be used as a basis for the analysis of nonlinear latent variable models. In this chapter we show how to calculate exactly the latent posterior distribution for the FA model in the presence of missing data, and note that this solution implies that a different encoder network is required for each pattern of missingness (Section 4.2). We empirically compare the effectiveness of various approaches for performing inference in the presence of missing data on a data imputation task (Section 4.3). Overall, our intention is to investigate approaches to dealing

with missing data for FA models, and in doing so highlight issues that should be addressed when performing inference in nonlinear latent variable models with missing data.

4.1 Introduction

Latent variable models, like factor analysis and the variational autoencoder (VAE, Kingma and Welling 2014; Rezende, Mohamed, and Wierstra 2014) are a compelling approach to modelling structure in complex high-dimensional data such as images.

One important use case of such models is when some part of the observable data is missing, for instance when some part of an image is not observed. In this case we would like to use the latent variable model to "inpaint" the missing data. A more practical example is modelling 3D pointcloud data; we may have observations of an object from one viewpoint, and wish to make inferences about the whole 3D object.

As discussed in Section 2.3.3 the VAE is composed of two parts, an *encoder* (or recognition) network that predicts the distribution of the latent variables given the data, and a *decoder* (or generative) model that maps from the latent variables to the visible variables. However, if some part of the visible data is missing, how can we make use of the encoder network in order to handle the missing entries? One simple fix is to replace the missing value with a constant such as zero (see e.g. Pathak, Krähenbühl, Donahue, Darrell, and Efros 2016, Nazábal, Olmos, Ghahramani, and Valera 2018) or "mean imputation", i.e. filling in the missing value with the unconditional mean of that variable. However, these are not principled solutions, and do not maintain uncertainty about the missing values.

Rezende et al. (2014) have shown that one can construct a Markov chain Monte Carlo (MCMC) method to sample from the posterior over the missing data for a VAE, but this is an iterative technique that has the obvious disadvantage of taking a long time for the chain to converge.

This paper is concerned with inference for the latent variables given a particular pre-trained generative model. The issue of learning a model in the presence of

missing data is a different issue; for PCA a survey of methods is given e.g. in Dray and Josse (2015).

The non-linear encoder and decoder networks of the VAE make an exact analysis of missing-data inference techniques challenging. As such we focus on the factor analysis (FA) model as an example autoencoder (with linear encoder and decoder networks) for which analytic inference techniques can be derived. In this paper we show that for the FA model one can carry out exact (non-iterative) inference for the latent variables in a single feedforward pass. However, the parameters of the required linear feedforward model depend non-trivially on the missingness pattern of the data, requiring a different matrix inversion for each pattern. Below we consider three approximations to exact inference, including a "denoising encoder" approach to the problem, inspired by the denoising autoencoder of Vincent et al. (2008). Experiments compare the effectiveness of various approaches to filling in the missing data.

4.2 Theory

We focus on generative models which have latent variables associated with each datapoint. We assume that such a model has been trained on fully visible data, and that the task is to perform inference of the latent variables in the presence of missing data, and to reconstruct the missing input data.

Consider a generative latent-variable model that consists of a prior distribution $p(\mathbf{z})$ over latent variables $\mathbf{z}$ of dimension K, and a conditional distribution over data variables given latents $p(\mathbf{x}|\mathbf{z})$. When there is missing data, the data variables $\mathbf{x}$ can be separated into the visible variables $\mathbf{x}_v$ and missing variables $\mathbf{x}_m$. $\mathbf{m}$ is a missing-data indicator function such that $m_j = 1$ if x_j has been observed, and $m_j = 0$ if x_j is missing. Let $\mathbf{x}$ have dimension D, with D_v visible and D_m missing variables.

Given a datapoint $[\mathbf{x}_v, \mathbf{m}]$ we would like to be able to obtain a latent posterior conditioned on the visible variables $p(\mathbf{z}|\mathbf{x}_v, \mathbf{m})$. We assume below that the missing data is *missing at random* (MAR, see Little and Rubin 1987 for further details), i.e. that the missingness mechanism does not depend on the value of the missing variables themselves.

One can carry out exact inference for linear subspace models such as factor analysis and its special case probabilistic principal components analysis (PPCA, Tipping and Bishop 1999). Let $p(\mathbf{z}) \sim N(0, \mathbf{I}_K)$ and $p(\mathbf{x}|\mathbf{z}) \sim N(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi})$ where $\mathbf{W}$ is the $D \times K$ factor loadings matrix, $\boldsymbol{\mu}$ is the mean (offset) vector in the data space, and $\boldsymbol{\Psi}$ is a D-dimensional diagonal covariance matrix. In these models the general form of the posterior is $p(\mathbf{z}|\mathbf{x}_v, \mathbf{m}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}_v}, \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v})$, where

$$\Sigma_{\mathbf{z}|\mathbf{x}_v} = \left(\mathbf{I}_K + \mathbf{W}_v^T \boldsymbol{\Psi}_v^{-1} \mathbf{W}_v\right)^{-1}, \tag{4.1}$$

$$\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}_v} = \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v} \mathbf{W}_v^T \boldsymbol{\Psi}_v^{-1} (\mathbf{x}_v - \boldsymbol{\mu}_v), \tag{4.2}$$

see e.g. Bishop (2007, sec. 12.2), where $\mathbf{W}_v$ denotes the submatrix of $\mathbf{W}$ relating to the visible variables, and similarly for $\boldsymbol{\Psi}_v$. These equations are simply the standard form for a FA model when the missing variables have been marginalized out. The expression for $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}$ can be re-written as

$$\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v} = (\mathbf{I}_K + \mathbf{W}^T \mathbf{M} \boldsymbol{\Psi}^{-1} \mathbf{M} \mathbf{W})^{-1}, \tag{4.3}$$

where $\mathbf{M}$ is a diagonal matrix the jth entry being m_j[1]. This means that the diagonal elements in $\mathbf{M}\boldsymbol{\Psi}^{-1}\mathbf{M}$ will be $m_j \psi_{jj}^{-1}$: if $m_j = 1$ (so x_j is visible) then x_j is observed with variance ψ_{jj}, but for missing data dimensions $m_j = 0$ implies an effective infinite variance for ψ_{jj}, meaning that any data value for this missing dimension will be ignored. Note also that an information-theoretic argument that conditioning on additional variables never increases entropy shows that $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}}$ will have a determinant no larger than $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}$[2].

Equation 4.3 can be rewritten in an interesting way. Let $\mathbf{v}_j^\intercal = (w_{j1}, \ldots, w_{jK})$ denote the jth *row* of $\mathbf{W}$. Then

$$\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}^{-1} = \mathbf{I}_K + \mathbf{W}^T \mathbf{M} \Psi^{-1} \mathbf{M} \mathbf{W} = \mathbf{I}_K + \sum_{j=1}^{D} m_j \psi_{jj}^{-1} \mathbf{v}_j \mathbf{v}_j^T, \tag{4.4}$$

i.e. where the second term on the RHS is a sum of rank-1 matrices. This arises from the fact that $p(\mathbf{z}|\mathbf{x}) \propto p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z}) \prod_{j=1}^{D} p(x_j|\mathbf{z})$, and can be related to the product of Gaussian experts construction (Williams and Agakov, 2002). Similarly

[1] Given that $\mathbf{M}$ is idempotent i.e. $\mathbf{M}^2 = \mathbf{M}$, one does not need to make use of the usual $\mathbf{M}^{1/2}$ construction on either side of $\boldsymbol{\Psi}^{-1}$.

[2] The entropy argument strictly applies when taking expectations over $\mathbf{x}$ or $\mathbf{x}_v$, but for Gaussian distributions the predictive uncertainty is independent of the value of $\mathbf{x}$ or $\mathbf{x}_v$, so it does imply the desired conclusion.

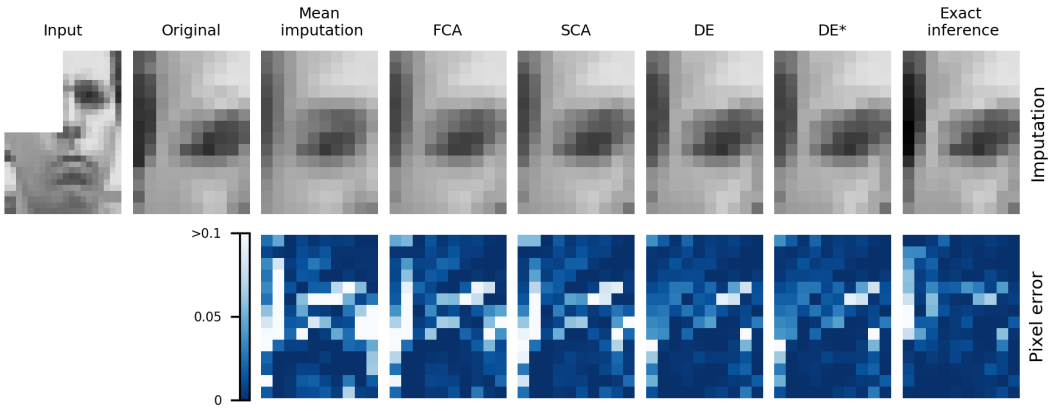


Figure 4.1: Illustration of the seven reconstruction methods on a specific image (leftmost) with the top left quarter missing. Top row: second column shows the original quarter, columns 3-7 show the corresponding imputations. The bottom row shows the pixelwise squared error.

the mean $\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}_v}$ has the form

$$\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}_v} = \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v} \sum_{j=1}^{D} m_j \psi_{jj}^{-1} (x_j - \mu_j) \mathbf{v}_j. \tag{4.5}$$

Again note how each observed dimension contributes one non-zero term to the sum on the RHS. The non-identifiability of the FA model due to "rotation of factors" (see e.g. Mardia, Kent, and Bibby 1979, sec. 9.6) means one can transform $\mathbf{W}$ with an orthogonal matrix $\mathbf{U}$ so that $p(\mathbf{z}|\mathbf{x})$ is a diagonal Gaussian. If $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, then setting $\tilde{\mathbf{W}} = \mathbf{W}\mathbf{U}$ makes the corresponding $\tilde{\boldsymbol{\Sigma}}$ be diagonal. Vedantam et al. (2018, sec. 2) suggest that for a VAE with missing data, one can combine together *diagonal* Gaussians in $\mathbf{z}$-space from each observed data dimension to obtain a posterior $p(\mathbf{z}|\mathbf{x}_v)$ using a product of Gaussian experts construction. However, our analysis above shows that this cannot be exact: even if we are in the basis where $p(\mathbf{z}|\mathbf{x})$ is a diagonal Gaussian, we note from eq. 4.4 that the contribution of each observed dimension is a rank-1 update to $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}^{-1}$, and thus there will in general be no basis in which all of these updates will be diagonal. However, note that a rank-1 update involves the same number of entries (K) as a diagonal update.

If the $\mathbf{x}$ data does not contain missing values, it is straightforward to build a "recognition network" that predicts $\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}}$ and $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}}$ from $\mathbf{x}$ using only matrix-vector multiplies, as per equations 4.1 and 4.2 without the v subscripts, as the matrix inverse can be computed once and stored. However, if there is missing data

Figure 4.2: Figure illustrating how the posterior means and standard deviations for unobserved pixels change as the amount of missing data decreases. Data above the red line is present, below it is missing and is imputed with exact inference.

then there is a different $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}$ for each of the $2^D - 1$ non-trivial patterns of missingness, and thus exact inference cannot be carried out efficiently in a single feedforward inference network. It is therefore desirable to consider more scalable approximations to the true posterior in the presence of missing data. Here we a present a number of approximation methods:

Approximation 1: Full Covariance Approximation (FCA). If we carry out mean imputation for $\mathbf{x}_m$ and replace those entries with $\boldsymbol{\mu}_m$ to give $\mathbf{x}^{mi}$, then we can rewrite eq. 4.2 as

$$\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}_v} = \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x}^{mi} - \boldsymbol{\mu}), \tag{4.6}$$

as the missing data slots in $\mathbf{x}^{mi} - \boldsymbol{\mu}$ will be filled in by zeros and have no effect on the calculation. However, note that $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}$ *does* depend on the pattern of missingness $\mathbf{m}$ as per eq. 4.3. A simple approximation is to replace it with $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}}$ which would apply when $\mathbf{x}$ is fully observed; we term this the *full covariance approximation*, as it uses the covariance relating to a *fully observed* $\mathbf{x}$. We denote this approximation as $\boldsymbol{\mu}^{FCA}_{\mathbf{z}|\mathbf{x}_v}$, as written in eq. 4.7:

$$\boldsymbol{\mu}^{FCA}_{\mathbf{z}|\mathbf{x}_v} = \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}}\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x}^{mi} - \boldsymbol{\mu}). \tag{4.7}$$

Approximation 2: Scaled Covariance Approximation (SCA). When no data is observed, we have that the posterior covariance is equal to the prior, $\mathbf{I}_K$. When there is complete data, then

$$\boldsymbol{\Sigma}^{-1}_{\mathbf{z}|\mathbf{x}} := \mathbf{P}_{\mathbf{z}|\mathbf{x}} = \mathbf{I}_K + \sum_{j=1}^{D}\mathbf{P}_j \tag{4.8}$$

where $\mathbf{P}_j = \psi_{jj}^{-1}\mathbf{v}_j\mathbf{v}_j^\intercal$, making use of eq. 4.4. Although as we have seen the $\mathbf{P}_j$s are all different, a simple approximation would be to assume they are all equal, and thus rearrange eq. 4.8 to obtain $\tilde{\mathbf{P}}_j = (\mathbf{P}_{\mathbf{z}|\mathbf{x}} - \mathbf{I}_K)/D$ for $j = 1, \ldots, D$.[3] This would give the approximation

$$\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}_v}^{-1} \simeq \mathbf{I}_K + \frac{\sum_j m_j}{D}(\mathbf{P}_{\mathbf{z}|\mathbf{x}} - \mathbf{I}_K) = \frac{D_m}{D}\mathbf{I}_K + \frac{D_v}{D}\mathbf{P}_{\mathbf{z}|\mathbf{x}}, \tag{4.9}$$

which linearly interpolates between $\mathbf{I}_K$ and $\mathbf{P}_{\mathbf{z}|\mathbf{x}}$ depending on the amount of missing data. Using this approximation instead of $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}}$ in eq. 4.7 yields the *scaled covariance approximation* (SCA). The inversion of the RHS of eq. 4.9 can be carried out simply and analytically if $\mathbf{P}_{\mathbf{z}|\mathbf{x}}$ is diagonal, as per the rotation-of-factors discussion above.

Approximation 3: Denoising Encoder (DE). Vincent et al. (2008) introduced the *denoising autoencoder* (DAE), where the goal is to train an autoencoder to take a corrupted data point as input, and predict the original, uncorrupted data point as its output. In Vincent et al. (2008) the stochastic corruption process sets a randomly-chosen set of the inputs to zero; the goal is then to reconstruct the corrupted inputs from the non-corrupted ones. As we have seen above, if the data is centered, then setting input values to zero is equivalent to mean imputation. Despite this, the motivation for denoising autoencoders was not so much to handle missing data, but to more robustly learn about the structure of the underlying data manifold. Also, as seen from our results above, a simple feedforward net that ignores the pattern of missingness (as in the DAE) cannot perform exact inference even for the factor analysis case.

However, we can develop ideas relating to the DAE to create a *denoising encoder* (DE). In our situation we have complete training data (i.e. without missingness). Thus for each $\mathbf{x}$ we can obtain the corresponding posterior for $\mathbf{z}$ (as per eqs. 4.1 and 4.2 without any missing data). We then create a pattern of missingness and apply it to $\mathbf{x}$ to obtain $\mathbf{x}^{mi}$. One can then train a regression model from $\mathbf{x}^{mi}$ to the corresponding posterior mean of $\mathbf{z}$. Note that this *averages* over the patterns of missingness, rather than handling each one separately. The *decoder* for the DE is the exact solution $p(\mathbf{x}|\mathbf{z}) \sim N(W\mathbf{z} + \boldsymbol{\mu}, \Psi)$. A limitation of the DE is that to

[3] A slightly more general decomposition would be to set $\tilde{\mathbf{P}}_j = \alpha_j(\mathbf{P}_{\mathbf{z}|\mathbf{x}} - \mathbf{I}_K)$ with $\sum_j \alpha_j = 1$, but then it would be more difficult to decide how to set the α_js; this could e.g. be learned for a given missingness mechanism.

train it we need examples of the patterns of missingness that occur in the data, which is not the case for the exact or approximate methods described above.

In future work it would be of interest to investigate the use of learned encoders that condition on both the visible data, as well as an input mask describing the presence of absence of the variables. Such a network would amortize the cost of inference across different patterns of missingness, and given a suitable training distribution of missingness patterns, may generalize effectively. An additional advantage of this approach is that an expressive posterior family could be used, and trained effectively as a conditional generative model with maximum likelihood, , enabling us to capture non-linear dependencies induced by the missing data. Such masks have been used to train autoregressive models with arbitrary variable orderings (Uria et al., 2014), as well as to indicate tokens to be predicted in large scale natural language processing tasks (Devlin et al., 2019).

4.3 Experiments

We trained a PPCA model on the Frey faces dataset[4], which consists of 1965 frames of a greyscale video sequence with resolution 20×28 pixels. Pixel intensities were rescaled to lie between -1 and 1. We used 43 latent components for the model, explaining 90% of the data variability. 80% of the data frames were randomly selected as a training set, and the remainder were held out as a test set.

We estimate the parameters $\mathbf{W}, \boldsymbol{\mu}, \sigma^2$ of the model using maximum likelihood, and obtain the analytic solutions as in Tipping and Bishop (1999). In the case of the denoising encoder, we create one missingness-corrupted pattern $\mathbf{x}^{mi}$ for each training example when estimating the regression model.

Given a trained PPCA model we investigate various approaches for handling missing data with a data imputation task. Given a partially observed data example, the task is to predict the values of the missing data variables using the trained model. We consider two patterns of missing data. In the first setting each data variable is independently dropped out with probability 0.5. In the second setting one of the four image quarters is dropped out in each example (see Fig. 4.1).

[4]Available from `https://cs.nyu.edu/~roweis/data/frey_rawface.mat`.

Table 4.1: Mean squared imputation error ($\times 10^2$) with standard errors for various imputation methods on PPCA generated data with random (R) and quarters (Q) patterns of missingness. For a particular data example the squared imputation error is the average error over imputed values for that example. Denoising encoder* is trained on random patterns of missingness, whereas Denoising encoder is trained on the same type of missingness patterns as it is tested on. For the Denoising encoder and Denoising encoder* on the Random data, note that the performance is identical as the training and testing data are the same in each case. This is indicated with brackets.

Approach	Random	Quarters
Mean imputation	4.5323 ± 0.0047	4.4642 ± 0.0065
Full-covariance approx.	1.8675 ± 0.0017	2.5732 ± 0.0045
Scaled-covariance approx.	0.9783 ± 0.0007	2.2671 ± 0.0041
Denoising encoder	1.0330 ± 0.0006	1.0043 ± 0.0011
Denoising encoder*	(1.0330 ± 0.0006)	1.6863 ± 0.0021
Exact inference	0.5913 ± 0.0001	0.6998 ± 0.0005

Table 4.2: Mean squared imputation error ($\times 10^2$) with standard errors for various imputation methods on the real Frey faces dataset. The details are as in Table 4.1.

	Random		Quarters	
Approach	Train	Test	Train	Test
Mean imputation	4.6390 ± 0.0013	4.6218 ± 0.0054	4.6002 ± 0.0017	4.4886 ± 0.0066
Full-covariance approx.	1.9339 ± 0.0006	1.9531 ± 0.0026	2.7613 ± 0.0012	2.7124 ± 0.0044
Scaled-covariance approx.	1.0433 ± 0.0004	1.0878 ± 0.0018	2.4820 ± 0.0011	2.4496 ± 0.0040
Denoising encoder	0.7161 ± 0.0002	1.1450 ± 0.0022	0.7262 ± 0.0003	1.2220 ± 0.0036
Denoising encoder*	(0.7161 ± 0.0002)	(1.1450 ± 0.0022)	1.8731 ± 0.0009	1.8883 ± 0.0036
Exact inference	0.6583 ± 0.0002	0.7165 ± 0.0016	1.2507 ± 0.0007	1.3677 ± 0.0035

We consider five approaches to handling missing data. The first method is simply to inpaint the missing variables with their training set means, without using the latent variable model at all. This is denoted by "mean imputation". The second approach is to use FCA, and the third SCA. For the fourth method we train a denoising encoder as described above, using a linear regression model. The final method is to use exact inference as per eqs. 4.1 and 4.2.

4.3.1 Results

We present two sets of results, firstly for data generated from the PPCA model fitted to the Frey data (denoted PPCA-Frey data), and secondly for the Frey data itself.

For the **PPCA-Frey data**, the mean squared imputation error results are shown in Table 4.1. In this case 400 test cases are generated from the PPCA model. For the DE case, 1600 training cases are also generated to train the DE model. For the denoising autoencoder on the quarters data, we train it either on data with missing quarters, or on data with pixels with a random pattern of missingness—the latter is denoted as Denoising encoder* in the tables.

In Table 4.1 we see that FCA improves over mean imputation, and that SCA improves over FCA. Training the denoising autoencoder (which uses the same input $\mathbf{x}^{mi}$ as FCA and SCA) gives a further improvement for the quarters data, but is very similar to SCA for the random missingness pattern. Note that on the quarters data, the Denoising encoder* results are quite a lot worse than the DE which had been trained on missing-quarters training data. For both the random and quarters data, the exact inference method is the best, as expected.

For the **Frey data** the mean-squared imputation error results are shown in Table 4.2, and a comparison on a specific image is shown in Figure 4.1. For the random missingness data, mean imputation is the worst and exact inference the best, with FCA, DE and SCA falling in between (from worst to best). Exact inference significantly outperforms the denoising encoder on the test set. For the quarters dataset, we see a similar pattern of performance, except for slightly better performance of the denoising encoder than exact inference on the test set. Notice also that when using the Denoising encoder* (training using random patterns of missingness) the performance is significantly worse. The slight performance gain of

the DE on the quarters data over exact inference may be due to the fact that exact inference applies to the PPCA model, but as the Frey data was not generated from this model there can be some room for improvement over the "exact" method. Also note that for the quarters data only four patterns of missingness are used, so the DE network is likely to be able to learn about this more easily than random missingness. There are noticeable differences between the training and test set errors for the DE model. This can be at least partially explained by noting that on the training set the DE model has access to the posterior for $\mathbf{z}$ based on the *complete* data, so it is not surprising that it does better here.

Figure 4.2 illustrates how the posterior means and standard deviations of unobserved pixels change as the amount of missing data decreases, using exact inference. As would be expected the uncertainty decreases as the amount of visible data increases, but notice how some regions like the mouth retain higher uncertainty until observed.

4.4 Conclusions

In this chapter we have discussed various approaches to handling missing data with probabilistic autoencoders. We have shown how to calculate exactly the latent posterior distribution for the factor analysis (FA) model in the presence of missing data. This solution exhibits a non-trivial dependence on the pattern of missingness, requiring a different matrix inversion for each pattern. We have also described three approximate inference methods (FCA, SCA and DE). Our experimental results show the relative effectiveness of these methods on the Frey faces dataset with random or structured missingness. A limitation of the DE method is that the structure of the missingness needs to be known at training time—our results demonstrate that performance deteriorates markedly when the missingness assumptions at train and test time differ.

Future directions could include the extension of our analysis to nonlinear models, and the development of methods to handle missing data in VAEs and other nonlinear models. As discussed in Section 4.2, training encoders conditioned on binary masks is a promising future direction, although our results for the denoising indicate a potential challenge in generalizing to out-of-distribution missingness

patterns. In recent work, steps have been made in this direction, with VAEs that augment the encoders and decoder with missing data indicator variables demonstrate good performance in data imputation tasks (Collier et al., 2020; Ma et al., 2019). A more straightforward alternative to this approach that is worth investigating, is to use three-way multiplicative interactions to modulate a linear transformation of the visible variables (Memisevic and Hinton, 2007). A low rank approximation to the cubic interaction matrix could then be employed to reduce the number of parameters required as in Memisevic and Hinton (2010).

Chapter 5

The Multi-Entity Variational Autoencoder

This chapter is adapted from the paper "*The multi-entity variational autoencoder*", presented at the 2017 NeurIPS 'Learning Disentangled Features' workshop.

We present an approach for learning probabilistic, object-based representations from data, called the multi-entity variational autoencoder (MVAE), whose prior and posterior distributions are defined over a set of random vectors. We describe the MVAE generative model, including the image decoder, and the maximal information attention mechanism (Section 5.2). In addition, we present experiments demonstrating the capacity of MVAE for between and within-object disentangling (Section 5.4).

5.1 Introduction

Human intelligence is object-oriented. Infants begin parsing the world into distinct objects within their first months of life (Spelke and Kinzler, 2007), and our sophisticated capacities for reasoning, imagining, planning, and learning depend crucially on our representation of the world as dynamic objects and their relations. Though the human notion of an object is rich, and exists in an even richer continuum of non-solids, non-rigids, object parts, and multi-object configurations, here we use the term "object" simply as a discrete visual entity localized in space.

Many important domains of artificial intelligence use representations of objects that were chosen ahead of time by humans, based on subjective knowledge of what constitutes an object (e.g. patches in images that can be categorized, or geometric meshes for physical control). This core object knowledge was learned through evolution and experience, and is very useful. It can be shared across object instances, provides a means for some properties of the world to be highly dependent and others to be relatively independent, and allows objects to be composed to form abstractions and hierarchies whose wholes are greater than the sums of their parts. Given the importance of such representations, and the high cost of manually translating it from the engineer's mind into AI datasets and architectures, this work asks: How can an artificial system learn, without supervision, an object-based representation?

Our contributions are: (1) a probabilistic model that can learn object-based representations from data, (2) a visual attention mechanism for inferring a sparse set of objects from images.

5.2 Multi-entity VAE

The multi-entity VAE (MVAE) is a latent variable model of data $\mathbf{x}$ in which the latent space is factored into a set of K independent 'object' representations $\{\mathbf{z}_1, \ldots, \mathbf{z}_K\}$. The MVAE defines a generative process in which each $\mathbf{z}_k$ ($k = 1, ..., K$) is sampled independently from a prior distribution, $p(\mathbf{z})$, and data examples are sampled from a decoder distribution $p(\mathbf{x} \mid \{\mathbf{z}_1, \ldots, \mathbf{z}_K\}; \boldsymbol{\theta})$.

In our experiments, the MVAE model assumes $p(\mathbf{z}_n)$ is an E-dimensional Gaussian with zero mean and unit variance. The parameters of the conditional data distribution are obtained using a three-step deterministic decoding function, $\mathbf{f}_\theta$ with parameters $\boldsymbol{\theta}$, which first maps each latent object representation to a processed object representation using a shared function, aggregates the processed object representations together, and deterministically transforms the result into the parameters of a Bernoulli distribution over pixel values. Crucially, $\mathbf{f}_\theta$ is permutation invariant with respect to the ordering of the object representations, which encourages the model to learn object representations that are consistent and interchangeable.

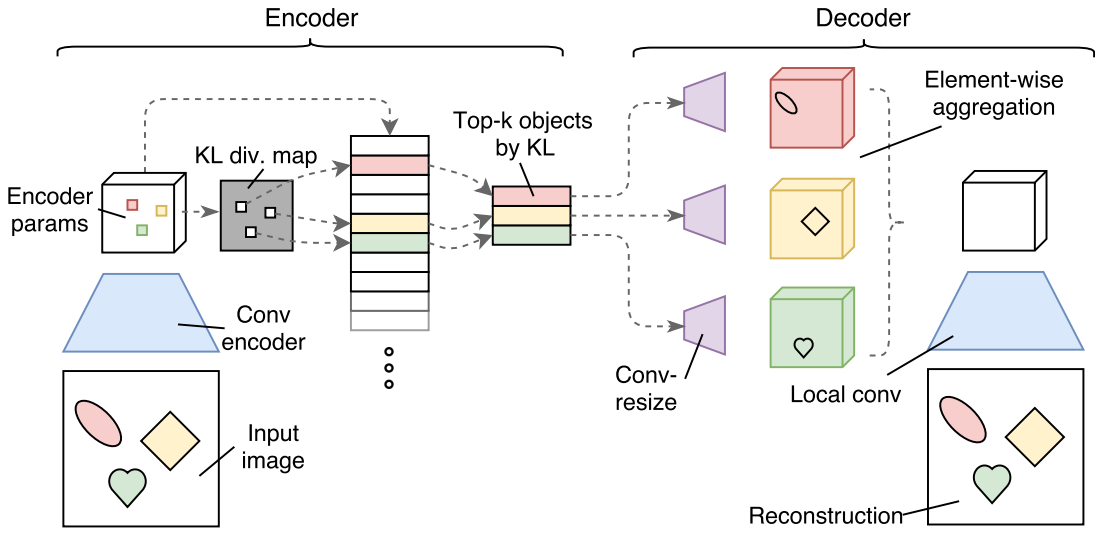


Figure 5.1: The Multi-entity VAE. The encoder takes input images and produces a spatial map of posterior parameters. The KL-divergence of the posterior distribution against the prior is computed in each spatial location. We use this as a proxy for the amount of information that needs to be encoded in each location to reconstruct the input image, represented as a KL-divergence spatial map. The K locations with the highest KL-divergence value are selected from the spatial map, and their associated posterior parameters are sampled. In the decoder these posterior samples are passed independently through a shared convolutional upsampling network. The resulting object feature maps are aggregated using an element-wise operation, and a final convolutional network produces the output parameters. At sampling time, K object vectors are sampled from the prior, rather than the posterior, and passed through the decoder network.

We use an attention-based mechanism to do inference over the latent object representations $\{\mathbf{z}_1, \ldots, \mathbf{z}_K\}$. The inference mechanism is a two-step process in which we first generate a super-set of candidate object inferences, and then sub-select the objects that contain the most information about the input scene. Figure 5.1 shows a schematic for the encoding and decoding processes.

Shared object processing. In the first stage of the decoder a shared function $\mathbf{g}_\theta$ is applied independently to each latent object representation, resulting in a set of spatially structured object descriptions $\mathbf{o}_k = \mathbf{g}_\theta(\mathbf{z}_k)$, $k = 1, \ldots, K$. These deterministic transformations of the prior latent variables are themselves random variables, which have dependencies induced by the prior latents. The object descriptions could be of any shape, but in this work we used 3D tensors as a

structural bias towards representations of visual attributes. We implement $\mathbf{g}_\theta$ as a network that transforms each latent vector to a 3D tensor via reshaping, convolutions and upsampling.

Aggregation. The processed object descriptions $\mathbf{o}_{1:K}$ are aggregated using a symmetric pooling function, to form $\mathbf{o}_{\text{pool}}$, a tensor with the same shape as each of $\mathbf{o}_{1:K}$. In our experiments we used element-wise sum or max as aggregation functions. For example, using max aggregation we have

$$o^{hwc}_{\text{pool}} = \max_{1 \leq k \leq K} \; o^{hwc}_k, \tag{5.1}$$

where h, w, and c index the vertical, horizontal, and channel dimensions of the spatial object representations.

Rendering. After pooling, the resulting $\mathbf{o}_{\text{pool}}$ is mapped (i.e. rendered) to the element-wise parameters of the decoder distribution using parameterized decoding function $\mathbf{h}_{\boldsymbol{\theta}}$. We use an independent Bernoulli distribution as the conditional pixel model, with a sigmoid cross-entropy reconstruction loss:

$$\mathbf{s}(\boldsymbol{\theta}, \mathbf{o}_{\text{pool}}) = \boldsymbol{\sigma}(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{o}_{\text{pool}})) \tag{5.2}$$

$$\mathcal{L}_{\text{rec}}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{o}_{\text{pool}}) = \mathbf{x} \log(\mathbf{s}(\boldsymbol{\theta}, \mathbf{o}_{\text{pool}})) + (1 - \mathbf{x}) \log(1 - \mathbf{s}(\boldsymbol{\theta}, \mathbf{o}_{\text{pool}})). \tag{5.3}$$

$\mathbf{h}_{\boldsymbol{\theta}}$ is a shallow, convolutional, upsampling network that takes in the spatially structured $\mathbf{o}_{\text{pool}}$ as input, and outputs pixel-wise Bernoulli logits. While the sigmoid cross-entropy does not correspond to a normalized distribution for the continuous pixel values, it is a widely used approach in the VAE literature (Sønderby et al., 2016; Dilokthanakul et al., 2016) that circumvents optimization challenges associated with scale-parameterized continuous distributions. In addition, we found it to work well in practice for the task of learning disentangled object representations. It would be of interest in future work to explore alternative, valid distributions on the continuous domain. For more discussion of this issues and a proposed solution in the form of a normalized 'continuous Bernoulli' see Loaiza-Ganem and Cunningham (2019).

5.2.1 Inference with maximal information attention

We employ amortized variational inference and learn a parameterized approximate posterior $q(\mathbf{z}_n|\mathbf{x})$ for each latent object representation. Unlike prior work (Eslami

et al., 2016; Gregor et al., 2015), we do not employ a learned attention mechanism in order to localise objects, but instead generate a large collection of candidate object inferences, from which K objects are selected. This inference method has the advantage that it circumvents the need for an explicitly learned attention mechanism, which may require a large number of recurrent passes over the image. This enables us to model scenes with large numbers of objects, something that was challenging in prior work.

Candidate generation. We generate candidate object inferences for visual scenes using a convolutional-network which maps input images to a grid of parameters $\boldsymbol{\mu}$, $\boldsymbol{\sigma}^2$ for a Gaussian distribution with diagonal covariance. Each spatial location s in this feature map of parameters is treated as a candidate object, and we sub-select from these candidates as described in the next section. After sub-selection the spatial structure present in the convolutional grid is destroyed, so we tag each object with its relative spatial coordinates at an intermediate feature map in the convolutional network.

Candidate sub-selection. Given a collection of candidate posteriors $\{q(\mathbf{z}_s|\mathbf{x})\}_s$ we compute the KL divergence $D^s_{\text{kl}} = D_{\text{kl}}[q(\mathbf{z}_s|\mathbf{x})|p(\mathbf{z})]$ for each candidate s. Approximate posteriors for the K latent objects are obtained by choosing the top-K object candidates by highest KL divergence value. The intuition for this process is as follows: In order to reconstruct the input the network must encode lots of information in locations where objects exist like their shapes, colours, etc., whereas much less information is needed to encode background information; simply that there is no object present there. As such the "object" and "non-object" locations will have high and low KL-divergence respectively, and by choosing the top locations by KL-divergence we encode information only in the most informative regions of the image. We call this process maximal-information attention, and note that it can be used for any data modality where a superset of candidate inferences can be generated.

The candidate generation and sub-selection results in approximate posteriors for the K object representations, which we then sample from and pass to the decoder as in a standard VAE. We train the MVAE using the VAE lower bound Kingma and Welling (2014), with the reconstruction log-probability term substituted with the sigmoid cross entropy loss as in Equation 5.3.

5.3 Related work

Our MVAE builds on previous neural probabilistic generative models, especially variational autoencoders (VAEs) (Kingma and Welling, 2014; Rezende et al., 2014). The DC-IGN (Kulkarni et al., 2015), beta-VAE (Higgins et al., 2017), and InfoGAN (Chen et al., 2016) are extensions and alternatives that promote learning latent codes whose individual features are "disentangled" (Tenenbaum and Freeman, 2000; DiCarlo and Cox, 2007), i.e. correlated exclusively with underlying axes in the data-generating process. Other work has developed attention mechanisms for sampling subsets of visual scenes (Mnih et al., 2014; Gregor et al., 2015; Vaswani et al., 2017), which promotes learned representations that are spatially localized. Recurrent neural networks have been used in combination with spatial attention to allow for unsupervised learning of object locations and counts Eslami et al. (2016). And several recent approaches allow object-like representations to be learned in supervised settings for visual question-answering and physical prediction (Santoro et al., 2017; Watters et al., 2017). Another related strand of work focuses on explicit representations of multiple objects in visual scenes, making use of graphics engines as a known decoding function (Wu et al., 2017; Romaszko et al., 2017).

5.4 Experiments

We evaluate our model on a multiple object variant of the dSprites dataset (Matthey et al., 2017). This dataset consists of 64×64 images of sprite shapes with random colours, orientations and locations as shown in Figure 5.2. dSprites is designed as a minimal dataset that tests a model's abilities to learn disentangled representations from pixel inputs, where the underlying factors of variation are known beforehand. In our experiments we aim to assess the extent to which our model learns distinct object representations. More concretely, we want to understand whether or not the MVAE's object 'slots' encode information about individual object sprites, or whether information about multiple objects is entangled in the same slot. We also aim to assess whether the MVAE is effective as a generative model on this data. Finally we assess whether the MVAE is effective at disentangling factors of variation *within* a particular object representation. Figure 5.3 shows training curves and Figure 5.4 shows model reconstructions and

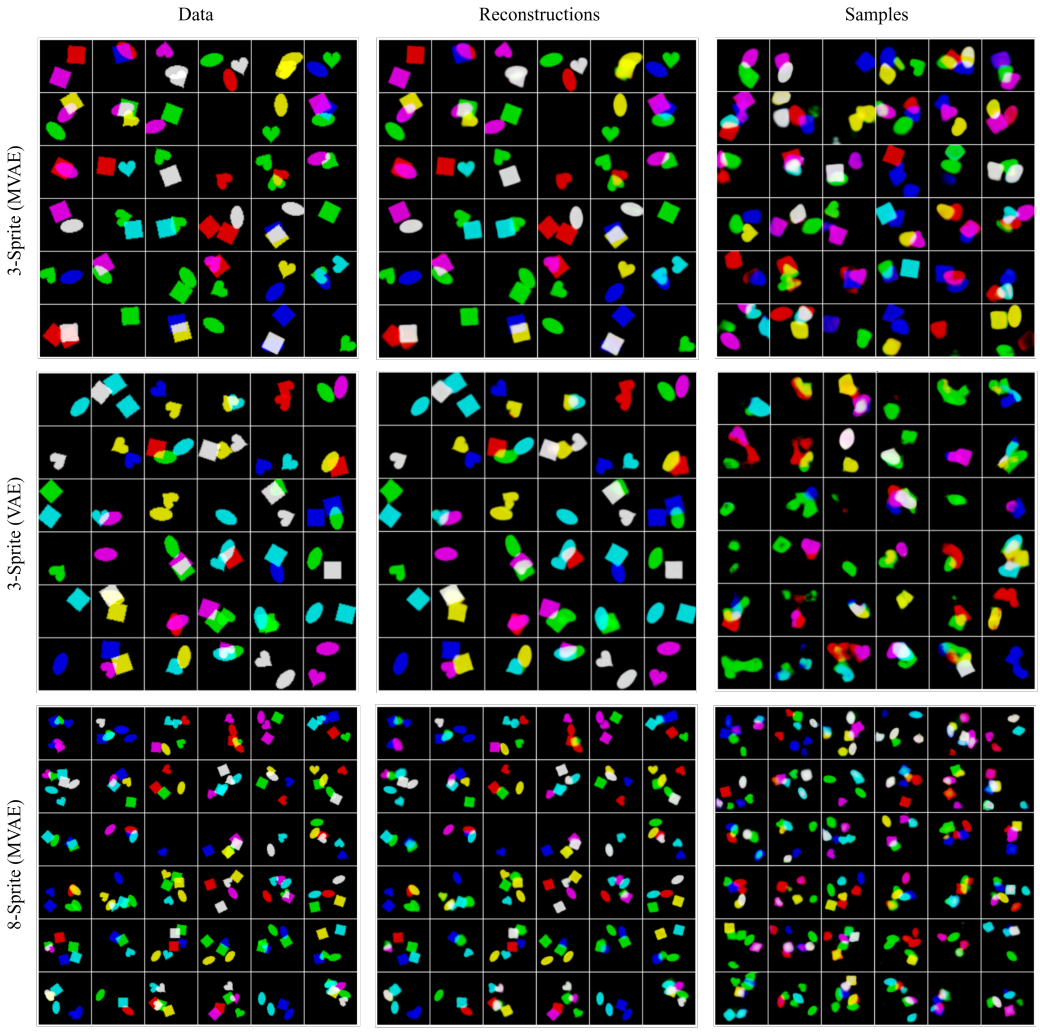


Figure 5.2: Reconstructions and samples. Data images, model reconstructions and model samples on the 3-sprite and 8-sprite datasets.

the development of KL-divergence attention maps over the course of training.

5.4.1 Experimental details

Data generation: Sprites are sampled independently, including whether or not they exist, up to some fixed upper bound on the number of sprites. We blend the colours of overlapping sprites by summing their RGB values and clipping at one. During training we generate these images dynamically, such that the MVAE always sees newly sampled data.

Encoder: The MVAE encoder is a convolutional network that starts with four convolutional layers with max-pooling. The resulting feature maps are

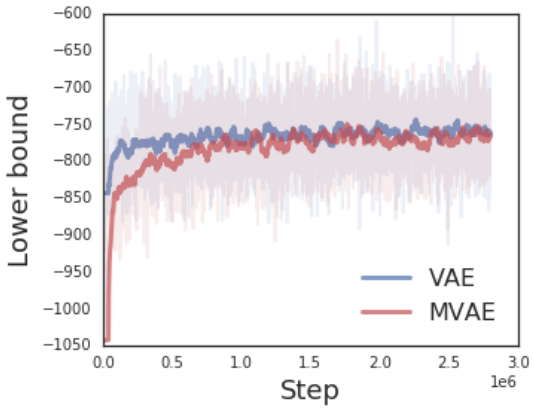


Figure 5.3: Training curves with exponential moving averages for the MVAE and a standard convolution VAE.

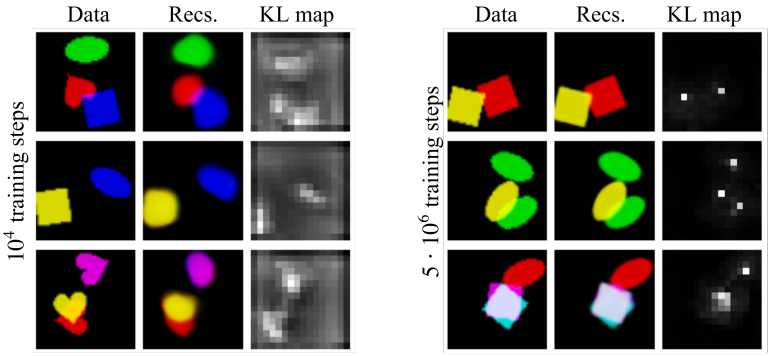
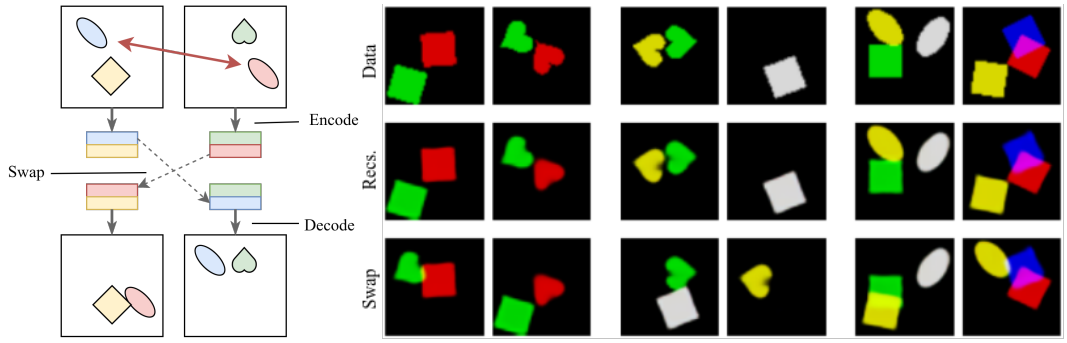


Figure 5.4: Model reconstructions and KL-divergence spatial maps at early and late stages of training. The KL maps are diffuse early in training and become increasingly sharp during optimization.

concatenated with two channels of relative x,y co-ordinates in order to encode the spatial locations of each position explicitly. Position-augmented feature maps are processed by two further convolutional layers with a final channel dimensionality of 24. The output channels are split to form the means and log-variances of 12-dimensional diagonal Gaussian posteriors.

Decoder: Latent object representations are processed with a shared network $\mathbf{g}_\theta$. This network has two fully connected layers, the outputs of which are reshaped into a feature maps of shape $8 \times 8 \times 16$, followed by two convolutional layers with bilinear up-sampling. We aggregate across object representations using max-pooling, and then process the resulting feature map with a convolutional network $\mathbf{h}_\theta$. This network consists of three convolutional layers and one bilinear up-sampling layer. The network outputs a $64 \times 64 \times 3$ image of Bernoulli logits.



(a) (b)

Figure 5.5: Entity exchange. (a) For a pair of input images, we encode to a set of latent objects before exchanging the representations of one objects in each pair. (b) Input pairs, model reconstructions, and reconstruction with exchanged entities. Here the objects in each input image with highest KL divergence are swapped.

Standard VAE: We trained a baseline VAE with a comparable architecture to the MVAE. We match the latent dimensionality, using 60 latent dimensions. The encoder is identical to the MVAE encoder, but the output feature maps are projected to the parameters of a 60-dimensional Gaussian using a linear layer. The decoder is equivalent to applying the shared object network $\mathbf{g}_\theta$ from the MVAE to the latent variables, and then passing the outputs into the rendering network $\mathbf{h}_\theta$.

Training: We train the MVAE using first-order gradient methods to maximize the variational lower bound on the log-probability of the data as in a standard VAE. All Models used elu non-linearities and were trained with Adam optimizer with scheduled learning rate annealing from 10^{-3} to 10^{-5} over the course of $3 * 10^6$ training steps.

5.4.2 Reconstructions and samples

Figure 5.2 shows reconstructions for 3-sprite and 8-sprite datasets. We note that in the 3-sprite data that both the VAE and MVAE achieve good reconstructions, however the MVAE samples are of a higher quality, with more distinctly defined and coherent objects. The MVAE achieves good reconstructions and samples for the highly cluttered scenes in the 8-sprite data.

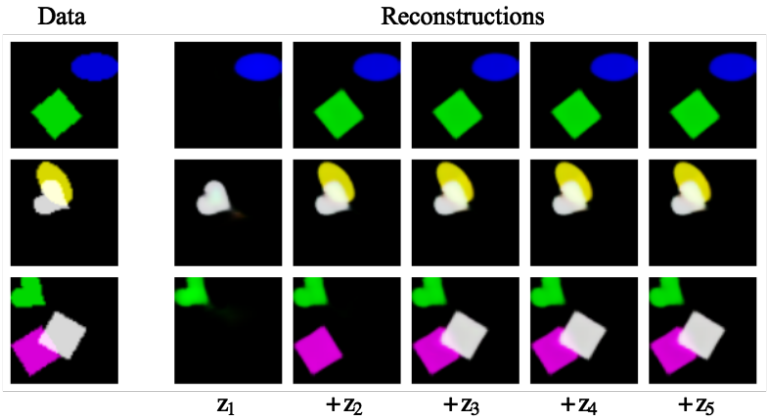


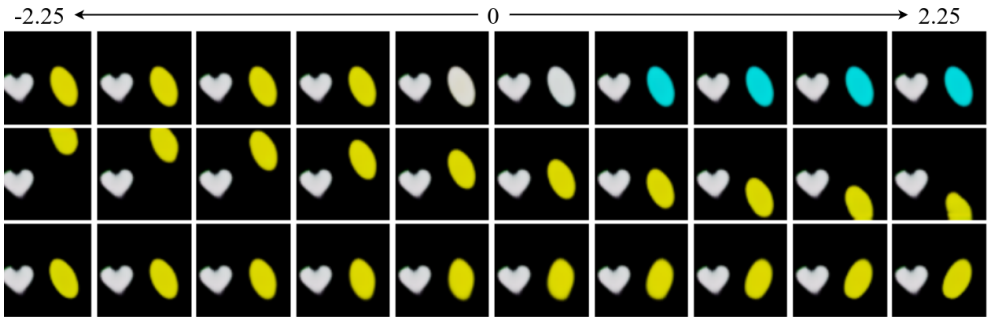
Figure 5.6: Decoding objects sequentially. (left) Data examples. (right) Reconstructions of the data where one latent object is added at each step. Here the latent objects $\mathbf{z}_1, \ldots, \mathbf{z}_5$ are ordered by the KL-divergence of the encoding distributions.

5.4.3 Between-object disentangling

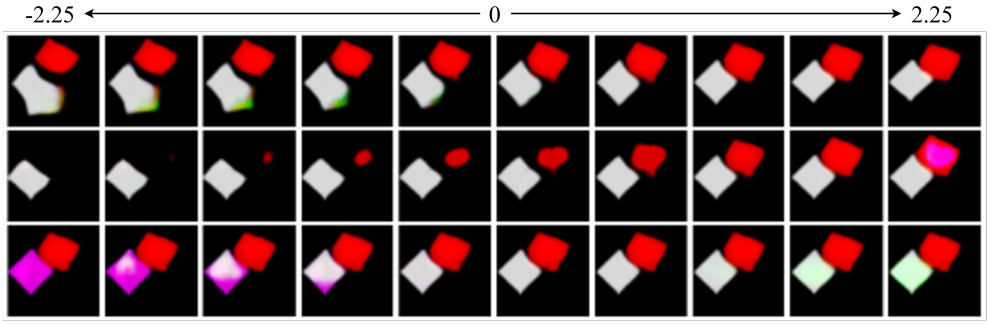
In order to check whether the MVAE has learned a factored representation of objects we use the following qualitative tasks:

Entity exchange. One of the main motivations for learning disentangled representations of objects in a scene is that it facilitates compositional reasoning. We should be able to imagine an object in different contexts, independent of the original context in which we observed it. We examine our model's capacity to transfer objects into new contexts by encoding a pair of scenes, swapping the representations for one object in each of the scenes, and then decoding both of the altered scenes. Figure 5.5 shows some example results for the MVAE. We note that entire objects are cleanly swapped between scenes, indicating that objects in the input image are cleanly partitioned into separate object representations.

Sequential decoding. Another qualitative indicator of the representations learned by the MVAE is to encode an input scene, then decode one object a time. As the MVAE's decoder performs object-wise aggregation, we can decode variable numbers of latent object representations. Figure 5.6 shows example decoded scenes in which we sequentially add one latent object representation at a time to the decoder. At each step a single object is introduced to the scene until all the objects present in the input scene have been decoded, and beyond this point the reconstructions are unaltered by the additional latent object representations. This indicates that the surplus latent object slots encode a 'Null' object, which decodes



(a) Multi-entity VAE



(b) Standard VAE

Figure 5.7: Decoded scenes for latent traversals from -2.25 to 2.25 for a selection of latent dimensions for a single object. The latent variables associated with different objects are held fixed during the process.

to nothing. We note that the individually decoded latents describe complete shapes, even in the presence of overlap in the inputs.

5.4.4 Within-object disentangling

To investigate the extent to which MVAE learns a representation that is disentangled within particular objects, we perform latent traversals for one object at a time, while keeping the other latent variables fixed. If the model has been successful we should expect to see that the underlying generative factors of the data are captured by single latent variables. Figure 5.7 shows a number of example latent traversals for MVAE and a standard VAE. The figure shows that the MVAE achieves a good degree of disentangling, e.g. with location factored from colour. This contrasts with the representations learned by a standard VAE, which are entangled across objects, with latent traversals causing multiple objects to deform and change colour simultaneously.

5.4.5 Unsupervised object counting

Here we demonstrate that the MVAEs spatial KL-divergence maps are a good proxy for object counting. We searched over KL-divergence thresholds on a training set of size 6400, and using the best training threshold tested on a 12800 newly sampled data examples. The chosen threshold gets 74.4% and 72.6% object count accuracy on the training and test sets respectively.

5.5 Discussion

Here we introduced a probabilistic model for learning object-based representations of visual scenes, which performs efficient inference using a novel one-shot informational attention mechanism that scales to large numbers of objects. Our results showed that our MVAE model can learn discrete, self-contained, interchangeable representations of multiple objects in a scene. We also found that the learned latent features of each object representation formed a disentangled code that separated the underlying factors of variation.

One limitation of this work is that the latent objects are assumed to be independent, which is obviously inconsistent with the tremendous statistical structure among objects in real scenes. Future work will tackle the task of learning not just about objects, but about their relations, interactions, and hierarchies in a unsupervised setting.

This work opens new directions for learning efficient and useful representations of complex scenes that may benefit question-answering and image captioning systems, as well as reinforcement learning agents.

Chapter 6

Inverting Supervised Representations with Autoregressive Neural Density Models

This chapter is adapted from the paper "*Inverting supervised representations with autoregressive neural density models.*" (Nash et al., 2019), presented at AISTATS 2019.

We present a method for feature interpretation that makes use of recent advances in autoregressive density estimation models to invert model representations. We train generative inversion models to express a distribution over input features conditioned on intermediate model representations (Section 6.4). Insights into the invariances learned by supervised models can be gained by viewing samples from these inversion models. In addition, we can use these inversion models to estimate the mutual information between a model's inputs and its intermediate representations, thus quantifying the amount of information preserved by the network at different stages (Section 6.4.1). Using this method we examine the types of information preserved at different layers of convolutional neural networks (Section 6.5.1), and explore the invariances induced by different architectural choices (Section 6.5.2). Finally we show that our estimate of the mutual information between inputs and network layers initially increases and then decreases over

the course of training, supporting recent work by Shwartz-Ziv and Tishby (2017) on the information bottleneck theory of deep learning (Section 6.5.3).

6.1 Introduction

The representations learned in supervised models are task specific; they discard irrelevant input information and preserve features that are useful for characterizing their targets. This is the conventional wisdom taken for granted by many in the machine learning community (Dosovitskiy and Brox, 2016a; Mahendran and Vedaldi, 2015; Zeiler and Fergus, 2014). However the precise nature of what information is preserved across different layers of a neural network is generally unknown. A better understanding of this is desirable both for the interpretability of a particular network, and for the insights that can be gained for neural architecture design.

We expect supervised models to be invariant to certain transformations of the input data. For instance an effective image classifier should be invariant to translations of objects in the image, and such behaviour is encouraged through architectural choices like convolutions and pooling Goodfellow et al. (2016). As such we anticipate that the mapping from inputs to intermediate representations discards information, and that perfect recovery of the inputs is not possible. Recent work by Schwartz-Ziv and Tishby (2017) argues that compression of input data in network representations is a central reason for the success of deep models, particularly with respect to generalization performance. Despite this, attempts by Dosovitskiy and Brox (2016a) have been made to invert representations using reconstructions that are optimized to minimize pixel losses such as mean squared error. This leads to blurry reconstructions from higher-level representations. Perceptual losses using features from a pretrained convolutional network or adversarial discriminator networks significantly improve the visual quality of results (Dosovitskiy and Brox, 2016b; Johnson et al., 2016), but fail to characterize the variability present in the inverse mapping. We propose instead to express a distribution over the inputs conditioned on a network representation. By sampling from this conditional distribution we can visualize the types of inputs that map to a given representation.

In recent years there have been significant advances in neural generative models

of high-dimensional data (Goodfellow et al., 2014; Kingma and Welling, 2014; Rezende et al., 2014). *Autoregressive* density models decompose a joint distribution into products of conditionals. By leveraging domain-specific structure, impressive results have been achieved with neural autoregressive models of images (van den Oord et al., 2016c,b) and audio (van den Oord et al., 2016a). Unlike alternative generative models such as variational autoencoders (Kingma and Welling, 2014; Rezende et al., 2014) or generative adversarial networks (Goodfellow et al., 2014), autoregressive models yield an exact density. We show in Section 6.4.1 that this density can be used to estimate a lower bound on the mutual information between inputs and model representations, which is a useful metric for the analysis of neural networks. In contrast to other methods for mutual information estimation this estimate scales to high-dimensionality inputs and network features with complex dependencies. In addition autoregressive models are straightforward to train in comparison to other generative models, with a single optimization objective and have none of the instability associated with adversarial training. Autoregressive density models are therefore a strong choice for our desired goal of representation inversion.

In this work we present a method for the inversion of supervised representations that uses flexible autoregressive neural density models to express a distribution over inputs given an intermediate representation. We show how such models can be used to help understand how much and what kind of information is preserved at different hidden layers on a range of image datasets (Sec. 6.5.1). We use inversion models to visualise the invariances learned by classifiers with different architectures, and demonstrate advantages in interpretability compared to point-estimate approaches (Sec. 6.5.2). Finally we demonstrate that our estimate of the mutual information between inputs and intermediate representations initially increases before decreasing over the course of training, reproducing the results of Schwartz-Ziv and Tishby (2017) in the context of ReLU-convolutional networks (Sec. 6.5.3).

6.2 Related work

Our approach is related to many previous works on inverting neural networks. Although our approach has similar goals to optimization-based approaches to

network inversion (Linden and Kindermann, 1989; Lee and Kil, 1994; Lu et al., 1999; Mahendran and Vedaldi, 2015) it is most closely related to methods which make use of another neural network that is trained to invert the hidden states (Dosovitskiy and Brox, 2016a,b; Johnson et al., 2016; Huang et al., 2017; Darlow and Storkey, 2020). In another related work, Zeiler and Fergus (2014) invert individual features by explicitly reversing the filtering, pooling and rectification operations in convolutional networks. Our work is distinct in its use of an autoregressive model to express a distribution over a network's inputs.

Dosovitskiy and Brox (2016a) train an up-sampling convolutional network to map from a representation layer $\mathbf{h}$ to inputs $\hat{\mathbf{x}} = \mathbf{f}(\mathbf{h})$, and optimize the mean squared error with respect to the true inputs $\mathbb{E}_{\mathbf{x}}||\mathbf{x} - \mathbf{f}(\mathbf{h})||^2$. With this method reconstructions become increasingly blurry as the amount of information preserved by the network about the inputs decreases with successive layers. The level of blurriness is quite useful as an indication of the amount of information preserved by the network, however it is a coarse measure that doesn't demonstrate the variability of inputs consistent with a given representation. In addition, this approach is only appropriate for data in continuous spaces where mean squared error is a meaningful metric. Our method can be applied in any setting in which a distribution over inputs can be parameterized, such as language processing. Nonetheless, the broad findings in this work are consistent with ours, suggesting that it is increasingly challenging to reconstruct input images for the deeper layers of the network.

Johnson et al. (2016) augment a pixel loss with perceptual losses that make use of the feature space of a pre-trained classifier to invert VGG features. These additional constraints result in outputs that are visually appealing in comparison to simple pixel losses. Dosovitskiy and Brox (2016b) extend this approach with an adversarial loss that encourages reconstructed inputs to additionally "fool" a GAN discriminator. Although these approaches produce high quality outputs they are limited in that they produce a single image reconstruction for a given representation, rather than providing a distribution over plausible inputs consistent with the representation. Both of these works indicate that while lower levels of image classifiers encode much of an image's pixel-level detail, it tends to be challenging to recover precise shapes, colours, and textures from higher levels of the network. As discussed in Section 6.5.1, this is consistent with the results in

this study.

Our method is also similar to stacked generative adversarial networks (Huang et al., 2017), in which a series of GANs are used to map from higher to lower level representations of a pre-trained classifier. This model was presented primarily as a way to make use of supervised representations in order to improve sample quality. In order to avoid degenerate samples, an entropy loss was incorporated that encourages the auxiliary noise to be recoverable from samples. This loss provides a lower bound on the entropy of the conditional distributions, and results in diverse samples. However entropy maximization is not equivalent to maximizing the likelihood, and may result in poorly calibrated distributions.

Van den Oord et al. (2016b) use a conditional PixelCNN to generate images conditioned on portrait embeddings obtained from the top layer of a face-detection CNN trained with triplet loss on Flickr images. This is equivalent to our method, although in that case the emphasis was on portrait generation rather than analysis of the learned representations, and no conclusions were drawn about the extent to which information is encoded in network layers.

In concurrent work, Darlow and Storkey (2020) similarly train PixelCNN++ models to invert the representations of ResNet image classifiers (He et al., 2016). The architectures studied in that work present an interesting case for analysis, as the residual connections mean that information propagation is straightforward for the network to achieve. Despite this, we find a broad agreement between with our results, in particular that successive layers in deep network architectures increasingly discard information, and that we can observe successive and compression phases over the course of training as described in Shwartz-Ziv and Tishby (2017).

6.3 Background

6.3.1 Autoregressive neural density models

Neural density models use neural networks to describe parametric distributions p_θ over random variables $\mathbf{x}$. Autoregressive models decompose the joint distribution into a series of D conditionals $p_\theta(\mathbf{x}) = \prod_i p_\theta(x_i|\mathbf{x}_{1:i-1}) = \prod_i p(x_i|\boldsymbol{\theta}_i)$ where the parameters for the ith conditional distribution are the outputs of a network

$\boldsymbol{\theta}_i = \mathbf{f}(\mathbf{x}_{1:i-1})$ that takes the preceding variables as input. Density models are typically trained to maximize the likelihood with respect to samples from the true data distribution. Various neural density models have been proposed; from general purpose models (Uria et al., 2013; Germain et al., 2015; Papamakarios et al., 2017) to domain specific models for images (van den Oord et al., 2016c,b), text (Sundermeyer et al., 2012), and audio (van den Oord et al., 2016a). Many neural density models make use of architectures that parallelize the computation of the D conditional distributions through a single pass of a network. This enables efficient computation as well as parameter sharing across conditional distributions. In order to ensure that each conditional only depends on the preceding variables, architectural tools such as causal convolutions and masking are used.

Conditional density models aim to model a conditional distribution $p(\mathbf{x}|\mathbf{h})$ of data variables $\mathbf{x}$ given context $\mathbf{h}$. Typical examples include models of images conditioned on object classes, or speech models conditioned on speaker identity. Usually each conditional is allowed to depend on the context such that $p_\theta(\mathbf{x}|\mathbf{h}) = \prod_i p_\theta(x_i|\mathbf{x}_{1:i-1}, \mathbf{h})$. We make use of conditional density models in order to model the distribution over input data conditioned on supervised representations.

6.3.2 PixelCNN

The PixelCNN (van den Oord et al., 2016c) is an autoregressive neural density model for images that uses a convolutional neural network to parameterize conditional distributions for each sub-pixel in an image. Pixel values are sampled one at a time: from left to right and from top to bottom. Causality in the conditional distributions is maintained using masked convolutions that only allow connections from previously observed pixels. The PixelCNN and its variants (van den Oord et al., 2016c,b; Salimans et al., 2017) are powerful models of images, and at the time of this work were state of the art with respect to log-likelihood scores on natural images. The current state of the art are for image log-likelihoods are Transformer-based autoregressive image models (Child et al., 2019; Vaswani et al., 2017). In our experiments we make use of the PixelCNN++ (Salimans et al., 2017), which incorporates a number of changes to the original model including the use of an alternative mixture-based pixel likelihood, downsampling to increase receptive field sizes and short-cut connections. Conditioning information is incorporated

by regressing a context vector to biases which are added to intermediate feature maps. For full details see Salimans et al., (2017) and the implementation at `https://github.com/openai/pixel-cnn`.

6.3.3 Information theory and mutual information

A quantity of particular interest in the analysis of network representations is the **mutual information** $I(\mathbf{x}; \mathbf{h})$ between inputs $\mathbf{x}$ and a model representation $\mathbf{h}$. The mutual information represents the reduction in uncertainty about $\mathbf{x}$ that we obtain if we know $\mathbf{h}$, and can be thought of in this context as the amount of information preserved in the transformation $\mathbf{x} \rightarrow \mathbf{h}$. In this section we define the mutual information and related information-theoretic quantities. For extended coverage of these topics see Cover and Thomas (2006) or MacKay (2003).

The mutual information is based on a scalar measure of uncertainty for random variables called **entropy**. High entropy distributions are relatively flat, placing probability mass broadly across a range of values, whereas low entropy distributions are peaky, and may be focused around a small number of values. For a discrete distribution $p(\mathbf{x})$ with values in $\mathcal{X}$ the **entropy** is defined as:

$$H\left(p(\mathbf{x})\right) = -\sum_{\mathbf{x}_k \in \mathcal{X}} p(\mathbf{x}_k) \log\left(p(\mathbf{x}_k)\right) = -\mathbb{E}_{p(\mathbf{x})}\left[\log\left(p(\mathbf{x})\right)\right] \tag{6.1}$$

For natural logarithms entropy is reported in units of "nats". Note that the entropy is a function of a probability distribution, and we are using $p(\mathbf{x})$ here to communicate the distribution associated with the random variable $\mathbf{x}$. This is useful notation for when we need to characterize a distribution by the random variables involved, as we do frequently in this chapter. For a conditional distribution $p(\mathbf{x}|\mathbf{h})$ with $\mathbf{h}$ taking values in $\mathcal{H}$ the **conditional entropy** is defined as:

$$H\left(p(\mathbf{x}|\mathbf{h})\right) = -\sum_{\mathbf{x}_k \in \mathcal{X}} \sum_{\mathbf{h}_j \in \mathcal{H}} p(\mathbf{x}_k, \mathbf{h}_j) \log\left(p(\mathbf{x}_k|\mathbf{h}_j)\right) = -\mathbb{E}_{p(\mathbf{x},\mathbf{h})}\left[\log\left(p(\mathbf{x}|\mathbf{h})\right)\right], \tag{6.2}$$

where it is useful to note that the expectation is taken with respect to the joint distribution $p(\mathbf{x}, \mathbf{h})$ and not the conditional distribution $p(\mathbf{x}|\mathbf{h})$.

The **mutual information** between two variables $\mathbf{x}$ and $\mathbf{h}$ is defined as the

reduction in entropy in $\mathbf{x}$ if we know $\mathbf{h}$, or vice versa:

$$I(\mathbf{x};\mathbf{h}) = H\left(p(\mathbf{x})\right) - H\left(p(\mathbf{x}|\mathbf{h})\right) = H\left(p(\mathbf{h})\right) - H\left(p(\mathbf{h}|\mathbf{x})\right). \tag{6.3}$$

It is straightforward to show that the mutual information is always non-negative. Intuitively, knowing additional information about $\mathbf{h}$ can never decrease our knowledge about $\mathbf{x}$. This is the key quantity that we seek to estimate, in order to understand how much input information is preserved in neural network layers.

Another quantity we will use later is the **cross entropy** between two distributions. To define cross entropy we overload our notation, re-using H with two input distributions $p(\mathbf{x})$, $q(\mathbf{x})$:

$$H(p(\mathbf{x}), q(\mathbf{x})) = -\sum_{\mathbf{x}_k \in \mathcal{X}} p(\mathbf{x}_k) \log\left(q(\mathbf{x}_k)\right) = -\mathbb{E}_{p(\mathbf{x})}[\log\left(q(\mathbf{x})\right)]. \tag{6.4}$$

As with the conditional entropy the expectation for the **conditional cross entropy** for distributions $p(\mathbf{x}|\mathbf{h})$ and $q(\mathbf{x}|\mathbf{h})$ is taken with respect to the joint distribution $p(\mathbf{x}, \mathbf{h})$:

$$H\left(p(\mathbf{x}|\mathbf{h}), q(\mathbf{x}|\mathbf{h})\right) = -\sum_{\mathbf{x}_k \in \mathcal{X}} \sum_{\mathbf{h}_j \in \mathcal{H}} p(\mathbf{x}_k, \mathbf{h}_j) \log\left(q(\mathbf{x}_k|\mathbf{h}_j)\right) = -\mathbb{E}_{p(\mathbf{x},\mathbf{h})} \log\left(q(\mathbf{x}|\mathbf{h})\right). \tag{6.5}$$

In machine learning we often aim to minimize the **empirical cross entropy** between a parameterized model $p_\theta(\mathbf{x})$, and a data distribution $p(\mathbf{x})$. For a collection of samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ from $p(\mathbf{x})$, the empirical cross entropy is defined as:

$$H^*(p(\mathbf{x}), p_\theta(\mathbf{x})) = -\frac{1}{N} \sum_n \log\left(p_\theta(\mathbf{x}_n)\right), \tag{6.6}$$

which is exactly the negative log likelihood of the parameters $\boldsymbol{\theta}$. it is also common to minimize the empirical cross entropy of a predictive distribution $p_\theta(y|\mathbf{x})$, and the data distribution $p(y|\mathbf{x})$. For a collection of paired samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ from $p(\mathbf{x}, y)$, the empirical cross entropy is defined as:

$$H^*(p(y|\mathbf{x}), p_\theta(y|\mathbf{x})) = -\frac{1}{N} \sum_n \log\left(p_\theta(y_n|\mathbf{x}_n)\right), \tag{6.7}$$

which is the familiar 'cross-entropy loss' used in classification models (Goodfellow et al., 2016).

For continuous random variables there exist analogous quantities based on the differential (or continuous) entropy. The **differential entropy** h is defined as an integral over the domain $\mathcal{X}$ of the variable:

$$h(\mathbf{x}) = -\int_{\mathbf{x}_c \in \mathcal{X}} p(\mathbf{x}_c) \log\left(p(\mathbf{x}_c)\right) d\mathbf{x}_c = -\mathbb{E}_{p(\mathbf{x})} \log\left(p(\mathbf{x})\right). \tag{6.8}$$

6.3.4 Mutual information estimation in neural networks

For neural network inputs $\mathbf{x}$, and representations $\mathbf{h}$ we are generally unable to obtain the mutual information directly, as we don't have access to the true distributions $p(\mathbf{x})$, $p(\mathbf{h})$, $p(\mathbf{x}|\mathbf{h})$ or $p(\mathbf{h}|\mathbf{x})$. However, the mutual information can be approximated in various ways, which we review in this section. It is important to first consider the implications of working with discrete vs continuous probability distributions.

Discrete vs continuous entropy. For deterministic functions of continuous inputs the conditional distribution $p(\mathbf{h}|\mathbf{x})$ is degenerate and so the differential entropy $h(p(\mathbf{h}|\mathbf{x}))$ is negative infinity. For finite $h(p(\mathbf{h}))$ this implies that the continuous mutual information $h(p(\mathbf{h})) - h(p(\mathbf{h}|\mathbf{x}))$ is infinite. This poses a problem for the analysis of neural networks with continuous inputs, as network representations are typically a deterministic function of the inputs. In this work we deal exclusively with discrete image inputs, and can avoid this issue by using discrete entropy for which $H(\mathbf{h}|\mathbf{x})$ is zero rather than infinite. However, for models with continuous inputs care must be taken to either add noise or to discretize the continuous space. For a more detailed discussion of related issues see Saxe et al. (2018).

Density estimation. For networks operating with continuous input spaces one method for mutual information estimation is to add noise to network activations $\mathbf{h}^\epsilon = \mathbf{h} + \boldsymbol{\epsilon}$ and use a parametric or non-parametric model $p^*(\mathbf{h}^\epsilon)$ to estimate $p(\mathbf{h}^\epsilon)$. As $\mathbf{h}$ is a deterministic function of $\mathbf{x}$ the conditional differential entropy $h(\mathbf{h}^\epsilon|\mathbf{x})$ is simply equal to the differential entropy of the Gaussian noise $h(\boldsymbol{\epsilon})$. The approximate model $p^*(\mathbf{h}^\epsilon)$ can then be used to estimate the continuous cross entropy $h(p(\mathbf{h}^\epsilon), p^*(\mathbf{h}^\epsilon))$, which is an upper bound on the continuous entropy $h(p(\mathbf{h}^\epsilon))$. An upper bound on the continuous mutual information can then be

established as follows:

$$I(\mathbf{x}; \mathbf{h}^\epsilon) = h(p(\mathbf{h}^\epsilon)) - h(p(\boldsymbol{\epsilon})) \leq h(p(\mathbf{h}^\epsilon), p^*(\mathbf{h}^\epsilon)) - c, \tag{6.9}$$

where $c = h(\boldsymbol{\epsilon})$, and we use the fact that $h(p(\mathbf{h}^\epsilon), p^*(\mathbf{h}^\epsilon)) \geq h(\mathbf{h}^\epsilon)$. Kolchinksy & Tracey (2017); Kolchinksy et al. (2017) use a kernel density estimate (KDE) $p^*_{\text{KDE}}(\mathbf{h}^\epsilon)$, and compute the empirical cross entropy to approximate the above bound. The approximation suffers from two potential issues: The first is error due to the sample-based estimate of the cross entropy. This issue is mitigated somewhat by when using large datasets to estimate the cross entropy. The second issue is that the KDE may be quite a poor model of the data, which will result in a loose bound. We discuss this in Section 6.4, as the same issue applies to our model. However, the issue may be exacerbated with KDEs in particular, as the performance of these models deteriorate significantly in higher dimensions. Theis et al. (2016) show that even for a very large number of samples, kernel density methods greatly underestimate the true log-likelihood of simple models trained on 6×6 image patches. As such these methods are not appropriate for the high-dimensional feature spaces of large networks. A parametric estimate $p^*_\theta(\mathbf{h}^\epsilon)$ using e.g. neural autoregressive density models would potentially scale better to higher dimensions, although to our knowledge this hasn't been explored in previous work.

Non-parametric direct entropy estimation. Saxe et al. (2018) similarly obtain an approximate bound on the continuous mutual information by estimating the differential entropy of activations with additive noise $h(\mathbf{h}^\epsilon)$. They use the estimator of Kraskov et al. (2004) that uses the distances between nearest neighbours in a collection of samples. The differential entropy estimator is:

$$\hat{h}(\mathbf{h}) = \frac{D}{N} \sum\nolimits_i \log(r_i + \epsilon) + \frac{D}{2} \log(\pi) \tag{6.10}$$

$$- \log \Gamma(D/2 + 1) + \psi(N) - \psi(k), \tag{6.11}$$

where D is the dimensionality of $\mathbf{h}$, N is the number of samples, r_i is the distance between sample i and its kth nearest neighbour, Γ is the Gamma function, and ψ is the digamma function. As with the KDE-based approach this non-parametric estimate may be problematic for analysis of network layers with very many units.

Discretization. Shwartz-Ziv and Tishby (2017) discretize tanh activations in a fully-connected neural network each into 30 equally sized bins to form a discrete

empirical distribution $p(\mathbf{h}|\mathbf{x})$. In experiments with a known distribution of discrete inputs $p(\mathbf{x})$, they exactly compute the mutual information between the inputs and the discretized layer activations by averaging over settings of $\mathbf{x}$ and $\mathbf{h}$. Saxe et al. (2018) note that the networks of interest do not operate on the discretized values, and that the binning is used solely for mutual information calculations. Moreover, there are many possible ways of binning potentially unbounded activations such as ReLUs, and the choice can significantly impact the mutual information estimates.

Both the non-parametric, and discretization-based mutual information estimation methods described above have issues, and estimating the mutual information between high-dimensional variables with complex dependencies is a fundamentally challenging task. But there are some room for improvements. In the following section we describe our approach to mutual information estimation which makes use of powerful deep generative models. By restricting ourselves to the case of discrete network inputs, we can avoid adding noise to the network activations, and by using expressive models we can obtain an estimate of a tighter lower bound on the mutual information.

6.4 Inverting supervised representations

Previous approaches to representation inversion (Dosovitskiy and Brox, 2016b) optimize a parameterized inversion function $\mathbf{f}_\theta$ with respect to the mean squared error of an image and its reconstruction:

$$\mathcal{L}_\theta^{\text{MSE}} = \mathbb{E}_{p(\mathbf{x},\mathbf{h})}||\mathbf{x} - \mathbf{f}_\theta(\mathbf{h})||^2. \tag{6.12}$$

This use of a point estimate in order to invert an information-lossy transformation results in blurry reconstructions and does not provide information about the variability inherent in the inverse mapping. Our method instead minimizes the negative log-likelihood of a parameterized *inversion* model:

$$\mathcal{L}_\theta^{\text{NLL}} = -\mathbb{E}_{p(\mathbf{x},\mathbf{h})}[\log\, p_\theta(\mathbf{x}|\mathbf{h})]. \tag{6.13}$$

This enables us to query whether a given input is a good match for a particular representation. In addition we can sample our trained model to get a sense of the degree of constraint present in $\mathbf{h}$. Optimization of Equation 6.12 is equivalent to our proposed maximum likelihood criterion if the conditional distribution p_θ is a

Gaussian with spherical covariance. As such our method simply extends Equation 6.12 by using a more flexible class of conditional probability models. As we have chosen to focus on supervised models of images, we use a conditional variant of the PixelCNN++ as our inversion model. Although we do not explore it here, we note that our method is not tied to any particular density model, and that equivalent domain-appropriate models could be used for e.g. text classification or speech recognition.

6.4.1 Estimating bounds on the mutual information

Inversion models can be used to compute an upper bound on $H(\mathbf{x}|\mathbf{h})$ using the conditional cross entropy between $p(\mathbf{x}|\mathbf{h})$ and the inversion model distribution $p_\theta(\mathbf{x}|\mathbf{h})$:

$$H(p(\mathbf{x}|\mathbf{h}), p_\theta(\mathbf{x}|\mathbf{h})) = -\mathbb{E}_{p(\mathbf{x},\mathbf{h})}\left[\log\, p_\theta(\mathbf{x}|\mathbf{h})\right] \tag{6.14}$$

$$= H(p(\mathbf{x}|\mathbf{h})) + \mathbb{E}_{p(\mathbf{h})}[D_{\mathrm{KL}}[p(\mathbf{x}|\mathbf{h})||p_\theta(\mathbf{x}|\mathbf{h})]] \tag{6.15}$$

$$\geq H(p(\mathbf{x}|\mathbf{h})). \tag{6.16}$$

This enables us to bound the mutual information as follows:

$$I(\mathbf{x};\mathbf{h}) = H(p(\mathbf{x})) - H(p(\mathbf{x}|\mathbf{h})) \tag{6.17}$$

$$\geq H(p(\mathbf{x})) - H\left(p(\mathbf{x}|\mathbf{h}), p_\theta(\mathbf{x}|\mathbf{h})\right). \tag{6.18}$$

This is similar to the density estimation approach described in Section 6.3.4, however instead of estimating $p(\mathbf{h}^\epsilon)$ we are estimating $p(\mathbf{x}|\mathbf{h})$. The gap between the true conditional entropy and the conditional cross entropy is given by the KL divergence between the true conditional distribution $p(\mathbf{x}|\mathbf{h})$ and our approximating distribution $p_\theta(\mathbf{x}|\mathbf{h})$ averaged over $\mathbf{h}$. Therefore the stronger our density model, the better the approximation to the conditional entropy will be. We should be cautious with how we interpret these estimates, as even with a strong generative model, estimating the distribution of high-dimensional data is highly challenging, and in general we should expect the bound to not be tight. We should be particularly cautious when comparing mutual information estimates for different network layers, or at different stages in training, as there may be a systematic reason why the bound changes for as these vary. For instance, it may be the case that during training, input information that is not useful for the task becomes

increasingly squashed into smaller and smaller floating point values. In this scenario the information is still present in the network activations, but it may become harder for the PixelCNN to effectively condition on, and we would observe a decrease in our mutual information estimates over time.

In practice we use the empirical conditional cross entropy by averaging across T, a held-out test set of $(\mathbf{x}, \mathbf{h})$ pairs:

$$H^*\left(p(\mathbf{x}|\mathbf{h}), p_\theta(\mathbf{x}|\mathbf{h})\right) = -\frac{1}{|T|}\sum_{(\mathbf{x},\mathbf{h})\in T} \log\, p_\theta(\mathbf{x}|\mathbf{h}). \tag{6.19}$$

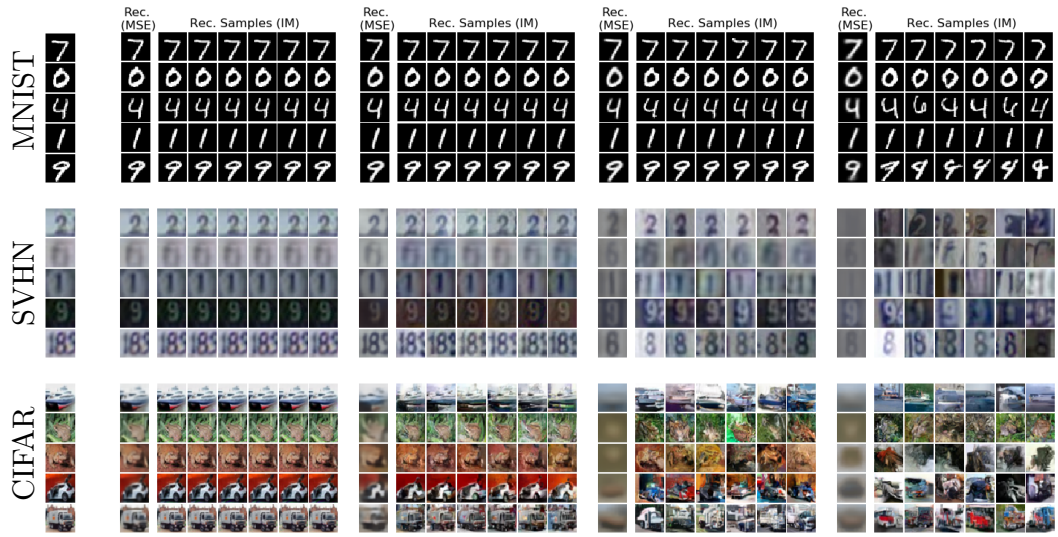
We could take this a step further, and directly estimate the mutual information by using an unconditional model $p_\theta(\mathbf{x})$ to estimate the marginal data distribution $p(\mathbf{x})$ and hence the entropy $H(p(\mathbf{x}))$. However, we don't typically need the absolute value of the mutual information for our analyses, but just the trends in how the mutual information changes between different network layers and settings. Thus, we directly report the estimated negative cross entropy (NCE), $-H^*\left(p(\mathbf{x}|\mathbf{h}), p_\theta(\mathbf{x}|\mathbf{h})\right)$, which is equivalent to the estimated mutual information bound, up to the constant $H(p(\mathbf{x}))$.

6.5 Experiments

6.5.1 Inverting the layers of image classifiers

We first explore the use of inversion models to explore the invariances and abstractions learned at each layer in a convolutional neural network.

Image classifiers. We trained classifiers on three image datasets: MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011) and CIFAR-10 (Krizhevsky and Hinton, 2009) achieving test accuracies of 99.6%, 93.3% and 81.6% respectively. Following Huang et al. (2017) we used ReLU-convolutional networks consisting of two convolutional layers with max pooling, one fully connected layer, and a linear layer that outputs the predicted logits for each class. We refer to these layers as CONV1, CONV2, FC3 and LOGITS respectively. For MNIST we used 32 filters in each convolutional layer, and for SVHN and CIFAR we used 64 and 128 filters for CONV1 and CONV2 respectively. The fully connected layer takes the vectorized feature maps of CONV2 and maps to FC3 which has 256 units for



(a) Input (b) CONV1 (c) CONV2 (d) FC3 (e) LOGITS

Figure 6.1: Inverting network layers. We compare both inversion model samples and MSE reconstructions across different layers of the neural network. For MNIST, even the logits retain much of the information about the original inputs. For SVHN and CIFAR the CONV1 and CONV2 layers appear to retain most of the information in the original images, while the FC3 layer becomes invariant to many low and mid-level changes.

all datasets. We used dropout at every layer, and Adam optimizer with learning rate 3×10^{-4}. We trained the networks for a maximum of 250000 steps, and used early stopping with respect to the validation accuracy.

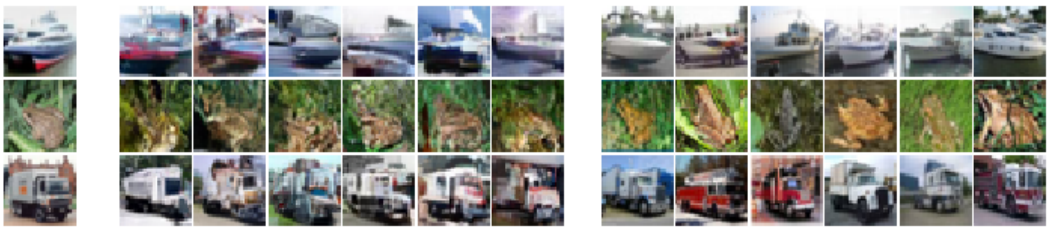
Inversion models. For each dataset and each classifier layer we trained separate PixelCNN++ inversion models. In each case we use the PixelCNN++ architecture detailed in Salimans et al. (2017). The architecture consists of six blocks of residual layers, with spatial downsampling using strided convolutions between the first, second and third blocks, and spatial upsampling using strided transpose convolutions between the fourth, fifth and sixth blocks. We attempted to preserve high resolution information with skip connections between corresponding downsampling and upsampling blocks. In order to reduce the cost of training the models, we used three residual layers in each block rather than the five specified in the original architecture. We use 64 filters in the convolutional layers for MNIST and 196 for SVHN and CIFAR. For SVHN and CIFAR we used the discretized mixture of logistics described in (Salimans et al., 2017) with 10 mixture components for the conditional pixel likelihood. For MNIST we used a 256-way softmax distribution

over the discrete pixel values as in the original PixelCNN (van den Oord et al., 2016c) as we found it to be much more effective in practice.

We used weight normalization with data-dependent initialization (Salimans and Kingma, 2016), and trained our models using Adam optimizer with initial learning rate 10^{-3} and a learning rate decay of 0.9999 for a maximum of 250000 weight updates. Again we used early stopping, but found that in general the validation performance continued to improve for the duration of training. To condition on vector representations FC3 and LOGITS we linearly projected the context vector to biases that are added to feature maps in each residual layer. For spatial representations CONV1 and CONV2 we resize the context to match the PixelCNN++ feature maps and use 1×1 convolutions to project to spatially-structured biases that are added to the feature maps. We used a single dropout layer in each PixelCNN++ residual block for all networks. For CONV1 inversion models on SVHN and CIFAR we used dropout rate 0.1, and for all other networks we used dropout rate 0.5. We applied an additional dropout layer with dropout rate 0.2 to the outputs of the linear projection of the context for the CIFAR-FC3 inversion model.

Samples. Figure 6.1 shows samples from trained inversion models along with reconstructions from models trained using the MSE loss from Equation 6.12. For SVHN and CIFAR the sample variability increases significantly from lower to higher layers. This is consistent with the increasing blurriness of the MSE reconstructions and confirms the expectation that input information is increasingly discarded in successive network layers. For MNIST the reconstructions are visually similar right up to the output layer, indicating that strong invariance in intermediate layers is not a requirement for good performance on this dataset.

For all datasets, CONV1 samples are almost indistinguishable from the inputs. CONV2 reconstructions also preserve information about the locations, styles and colors of objects and digits, but are more variable with respect to finer details. There is a distinct increase in reconstruction variability from CONV2 to FC3, particularly on the CIFAR dataset. However color information is preserved in FC3, along with object structures and scene textures. A surprising amount of information is retained even in the networks' logit predictions; this is particularly evident in the MNIST reconstructions for which style and orientation information is preserved. This is consistent with the dark knowledge hypothesis described by



(a) Input (b) Top-K inversions (c) Nearest neighbours

Figure 6.2: Comparing inversion model samples to nearest neighbors. For each input image (a) we sample 1024 images using an FC3 inversion model, and show the top-k by L_1 distance (b). We compare these to the top-k nearest neighbors from the training set calculated in the same way (c).

(a) MNIST (b) SVHN (c) CIFAR

Figure 6.3: SGAN samples conditioned on FC3. The SGAN samples are less diverse than the inversion model samples and the nearest neighbors shown in Figure 6.2. In the SGAN model, this diversity is controlled by an entropy loss term with a tunable weight, making it difficult to calibrate it in a principled way.

Hinton et al. (2015) that suggests that the particular output probabilities that a model assigns to its inputs provides a rich characterization of the similarity between examples.

Mutual information. Figure 6.5a shows estimates of lower bounds on the negative cross entropy at the different layers in the neural network, as discussed in section 6.4.1. In general, the mutual information bound estimates decrease through successive layers of the network, suggesting a general loss of information about the input. The biggest reduction in mutual information by far is between CONV1 and CONV2, which is surprising as the biggest jump in variability for samples from the inversion models appears to occur in later layers. It is possible that our intuitions about visual information are poorly calibrated, and that we underestimate the amount of information present in high-frequency pixel detail. This is supported by the popularity of perceptual loss metrics in generative vision applications (Johnson et al., 2016; Dosovitskiy and Brox, 2016b), that aim to characterize the differences between images in a feature space better aligned with human perception. As discussed in Section 6.4, it is possible that these

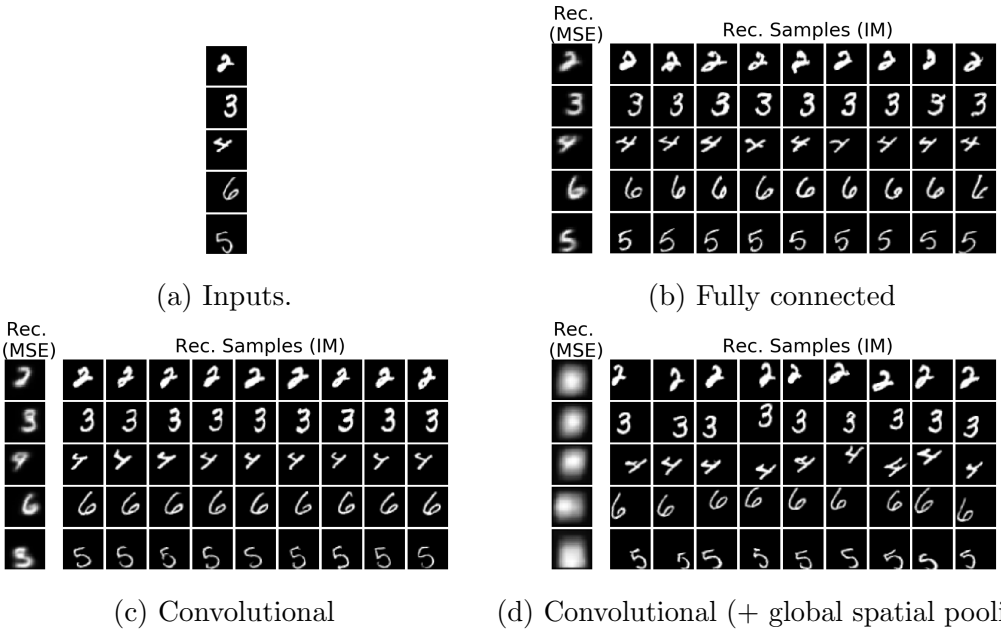
	MNIST			**SVHN**			**CIFAR**		
	CONV1	CONV2	FC3	CONV1	CONV2	FC3	CONV1	CONV2	FC3
1NN	1.25e-2	4.95e-2	1.22e-1	5.35e-2	6.74e-2	2.92e-1	4.11e-2	9.56e-2	**5.28e-1**
IM-S	7.68e-4	1.65e-2	1.35e-1	1.02e-3	2.80e-2	3.22e-1	4.02e-3	6.43e-2	7.39e-1
IM-NN	**6.29e-4**	**1.40e-2**	**9.52e-2**	**9.40e-4**	**2.43e-2**	**2.50e-1**	**3.47e-3**	**5.73e-2**	5.79e-1

Table 6.1: Comparing images and reconstructions in representation space. For 500 test samples from each dataset we compute the average L_1 distance at various network layers between the input image and: (1NN) the nearest neighbor training set example, (IM-S) a random inversion sample or (IM-NN) the closest of 10 inversion model samples. We can see that the IM-NN distance is the closest for all datasets and layers except the FC3 layer of CIFAR10.

observations do not correspond to a real change in mutual information, but rather to a change in the accessibility of the information to the PixelCNN.

Comparison to nearest neighbors and SGAN. In order to evaluate how well the inversion models capture the distribution $p(\mathbf{x}|\mathbf{h})$ we pass the generated inversion samples, $\hat{\mathbf{x}}$, back through the image classifier to generate the hidden states $\hat{\mathbf{h}}$ for these generated images. Table 6.1 shows the results of doing this for 500 test set images and computing the average L_1 distance between $\mathbf{h}$ and $\hat{\mathbf{h}}$. We also perform the same calculation using the nearest neighbor images in the training set, instead of generated samples. We report the distances for a single random sample from an inversion model (IM-S) as well as for the closest of 10 random samples (IM-NN). We find that for a single random sample, the FC3 L_1 distance is smaller than the that of the nearest training set example for CONV1 and CONV2, but not for FC3. The IM-NN distances are the smallest for all layers on the MNIST and SVHN datasets, but not on the CIFAR dataset. Figure 6.2 shows a selection of nearest-neighbour training examples, as well as the closest inversion model samples from a collection of 1024, for a selection of input images.

For comparison, in Figure 6.3 we also show samples from an SGAN conditioned on the FC3 layer of an equivalent CNN (reproduced from (Huang et al., 2017)). We can see that the variability of these samples is much lower than both the samples from our model and from the nearest neighbors in Figure 6.2c. In fact, in order to prevent the SGAN from collapsing to a deterministic function, the



(a) Inputs.

(b) Fully connected

(c) Convolutional

(d) Convolutional (+ global spatial pooling)

Figure 6.4: Inversion model samples and MSE reconstructions from the FC3 layers of three network architectures trained on affNIST. The network with global pooling learns translational invariance, while the others do not. Furthermore, the inversion samples indicate that the style and orientation of the digits are retained in the global pooling network, an observation that cannot be gleaned from the MSE reconstructions.

authors added an explicit entropy term to the loss function. As the weight on this loss term is a tunable hyperparameter, there's no principled way to set this to ensure that the variability is well calibrated. In contrast, since we are using an autoregressive model which is trained using maximum likelihood, we need no such tunable hyperparameter, and can expect the variance to be better calibrated. It's also worth noting that our use of autoregressive models enables estimation of the mutual information, as discussed above, a capability not provided by the SGAN model.

6.5.2 Comparing network architectures using inversion models

Another practical application of inversion models is to facilitate analysis of network architectures by revealing the invariances present in various network layers. If we know that our networks are not learning the desired invariances, we can take steps to modify our architecture. As a case study, we analyze a design choice for

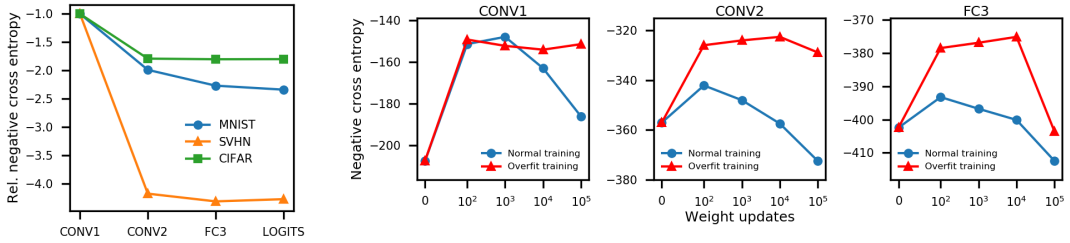
convolutional architectures that has become increasingly popular: global spatial pooling. Global pooling layers aggregate information from all spatial locations, greatly reducing the number of parameters in network architectures. They have been found to help reduce overfitting, and have largely replaced fully connected layers in the final processing steps in modern image architectures (Lin et al., 2014; He et al., 2016). As an additional point of comparison we include a network that replaces convolutional layers with fully connected layers.

We train supervised models on the affNIST dataset[1]. affNIST consists of MNIST digits with random affine transformations on a 40×40 canvas. These transformations increase the need for invariance in network representations in comparison to standard MNIST. We trained three supervised networks: the first is identical to that used on MNIST in Section 6.5.1, the second applies global max pooling to the CONV2 feature maps and passes the resulting vector to FC3. The final network replaces convolutional layers with fully connected layers with 2048 units. The network with global pooling performs best, achieving 98.9% accuracy compared to 98.7% for the version without global pooling and 95.0% for the fully connected network. We trained PixelCNN++ inversion models to invert FC3 representations for the supervised networks, using the same architecture as for the MNIST model in section 6.5.1.

For the fully connected network we estimate the relative mutual information lower bound to be -715.17 nats. For the convolutional networks with and without global pooling we observe -704.01 and -699.55 nats respectively. These estimates indicate that more input information is preserved in the layers of the convolutional networks than the fully-connected network, and that the global pooling layer discards an estimated ~ 4.5 nats of information about the inputs.

Figure 6.4 shows samples from the inversion models along with MSE reconstructions. The samples indicate that for the network with global pooling the translation of the digit is not preserved in FC3, whereas for the network without global pooling and the fully connected network the digit's location is preserved. These results are reinforced by the MSE reconstructions, which are very blurry for the network with global-pooling. It should be noted that it is not possible to tell from the MSE reconstructions that the global-pooling network preserves style and rotation information about the digit, whereas samples from the inversion model

[1]Available at `https://www.cs.toronto.edu/~tijmen/affNIST/`



(a) All Datasets (b) MNIST across training steps

Figure 6.5: Mutual information analysis. (a) Negative Cross Entropy (NCE) across datasets and layers, expressed relative to the magnitude of the CONV1 cross entropy. (b) Absolute NCE in nats for network layers over the course of training for regular (blue) and intentionally overfitted (red) classifiers on MNIST. The initial increase and subsequent decrease in NCE over the course of training for the regular model is consistent with the expansion and compression phases predicted in previous work (Shwartz-Ziv and Tishby, 2017). The NCE is considerably higher for the overfitted network.

indicate that this is the case. It is surprising that the models without global pooling do not achieve translation invariance by FC3, and it may explain their relatively poor performance, as the networks must learn this invariance in the final linear layer.

6.5.3 Training dynamics

Inversion models can also be used to better understand the compression dynamics of neural network training. These dynamics were recently studied by Schwartz-Ziv and Tishby (2017) who examined the mutual information between inputs and intermediate layers over the course of SGD optimization. Their findings suggested that there exist two distinct phases of training: an expansion phase in which networks increase the mutual information between inputs and hidden layers and a compression phase in which the mutual information is reduced as information is filtered out. More recently Saxe et al. (2018) provided evidence that the compression phase only occurs when saturating non-linearities such as tanhs are used, and that no compression is present for ReLU networks. In both cases, the mutual information was estimated using either discretization or non-parametric methods, each of which have issues in terms of scalability to high dimensionality features as discussed in Section 6.3.4. Instead, we can use inversion models to

examine these claims for the larger networks that we employ.

Using the MNIST classifier described in Section 6.5.1 we trained inversion models on representations established after 0, 10^2, 10^3, 10^4 and 10^5 weight updates. We use this relatively coarse view, rather than the finer view taken by Schwartz-Ziv and Tishby due to the computational expense of training inversion models. For the supervised MNIST network, which takes about 5×10^4 weight updates to converge, this range is fairly representative of the training process. In order to investigate the connections between mutual information and generalization we additionally trained an overfitted network by using only 100 training examples and removing dropout from the classifier. We use equivalent inversion models to the ones described in section 6.5.1.

Figure 6.5b shows estimates of the negative cross entropy of inversion models trained at different network layers over the course of training. As discussed in Section 6.4.1 The results for normal training are shown in blue, and the results for the overfitted network are shown in red. Our main observation is that in the normal training regime for all layers of the network the lower bound estimates of the mutual information initially increases, and then decreases significantly over the course of training. We therefore see a reproduction of the main findings of Schwartz-Ziv and Tishby for ReLU-convolutional networks. This apparent contradiction of the findings of Saxe et al. (2018) can potentially be explained by their use of non-parametric methods to estimate the mutual information. Our results additionally indicate that the mutual information is considerably higher for the overfitted network than for the well-regularized network, which supports the notion that compression in network layers has an important role in a model's generalization performance. However this comes with the caveat discussed in Section 6.4 that we can't rule out a systematic change in lower bound tightness between the overfitting and non-overfitting modes.

6.6 Discussion

In this work we present a method for the inversion of supervised representations that uses flexible autoregressive neural density models to express a distribution over inputs given an intermediate representation. Our method has two benefits: it

facilitates visualisation of model invariances, thus enabling analysis of architectural choices. Secondly it provides a scalable quantitative estimate of the amount of information preserved by a network. One difficulty is that density estimation is challenging in higher dimensions, and that we don't know how well a given model represents the true density. However, as neural density models improve, so too does our method, and we will be able to achieve tighter mutual information bounds and more representative samples.

There are a number of directions for future work, including the comparison of representations learned using different optimizers, or in different training regimes. Additionally it would be of interest to analyze the effect of training techniques such as dropout or batch normalization, or of architectural choices such as residual connections on representation and compression.

Chapter 7

Conclusions and Future Work

In this thesis we have explored neural latent variable models as a powerful tool for modelling high-dimensional data. By assuming an underlying low-dimensional structure, these models give us the capacity to capture complex non-linear dependencies between hundreds or thousands of variables. We have explored structured models like the multi-entity VAE (Chapter 5), where latent variables describe distinct objects in visual scenes, or the ShapeVAE (Chapter 3) where latent variables interact with discrete part variables to model the shape of diverse 3D objects. We have addressed challenges in performing inference over latent variables in the presence of missing data, whether in the case of missing parts with the ShapeVAE, or more generally for linear latent variable models (Chapter 4).

Clearly, VAE-style latent variables provide powerful modelling capabilities. However, they are not the only tool in the generative modelling toolbox. If we want to produce visually realistic samples, particularly in the image domain, then generative adversarial networks (Goodfellow et al., 2014; Brock et al., 2019; Karras et al., 2019) produce the best results. If we want to model data dependencies as effectively as possible, and we don't care about latent structure, then autoregressive models, as used in Chapter 6 achieve the strongest log-likelihood scores (van den Oord et al., 2016c; Uria et al., 2013). Normalizing flows similarly offer good density estimating performance, combined with more efficient sampling than autoregressive models (Dinh et al., 2017; Kingma and Dhariwal, 2018). As such, depending on our requirements, VAE-style latent variable models are not always the best choice of model.

But, if there really is some underlying latent structure that explains our observations, then latent variable models, and the tools of amortized inference and deep learning should be at the forefront of our mind. They open up an intuitive modelling paradigm where we can think about how the data is generated in terms of latent variables, while enabling deep networks to learn the nonlinear relationships that we can't specify beforehand.

7.1 Summary of contributions

In **Chapter 3** we presented the **shape variational autoencoder**, a deep latent variable model of part-structured 3D objects. The ShapeVAE describes a joint distribution over thousands of 3D point positions, surface normals and part existence variables, and can generate plausible and diverse shapes. By generating surface normals in addition to surface points, the ShapeVAE allows for the use of normal-based mesh reconstruction methods, improving the quality of mesh reconstructions. The model performs favourably relative to strong baselines in terms of test log-likelihood and shape completion tasks, and we provide a qualitative demonstration of the latent shape representation learned by the model.

In **Chapter 4** we analyzed methods for latent variable inference for linear subspace models in the presence of **missing data**. In particular, we show that the exact posterior distribution depends on which input variables are present, requiring a distinct matrix inversion for each setting. In addition we present an empirical analysis of the performance of a number of approximate inference methods on the Frey faces dataset with random and structured missing data.

In **Chapter 5** we present the **multi-entity VAE**, a latent variable model of visual scenes that decodes a collection of latent object vectors. We introduce an information-based attention mechanism that enables one-shot inference of object attributes, that scales to large numbers of objects. We also found that the learned latent features of each object representation formed a disentangled code that separated the underlying factors of variation, such as colour, shape and pose.

In **Chapter 6** we describe a method for the analysis of neural network features that uses autoregressive decoders called **inversion models** to express a distribution over input features conditioned on intermediate model representations. We show

how inversion models can be used to gain insights into the invariances learned by supervised models over the course of training. In addition, they can be used to estimate the mutual information between a models inputs and its intermediate representations, enabling us to quantify the amount of information preserved by the network at different stages. Using this method we examine the types of information preserved at different layers of convolutional neural networks, and explore the invariances induced by different architectural choices.

7.2 Future work

The work in this thesis opens up a number of avenues for further research. In this section we discuss these areas in the context of recent developments in the field.

The ShapeVAE presented in Chapter 3 is effective at capturing shape variability for complex shape classes, but there is an important caveat: The model assumes that we can establish correspondences between points on different objects within a class. However, in general such correspondences may not be available or meaningful, due to topological or structural differences between examples. It is desirable to relax this constraint, and to learn generative shape models on more workable shape representations. Since the ShapeVAE was published in 2017, there have been some promising steps in this direction. For instance, there now exist models that operate on unordered point-clouds, making use of set-to-set metrics like chamfer distance as a learning objective (Fan et al., 2017; Groueix et al., 2018). There has been promising progress using alternative shape representations like voxels (Tatarchenko et al., 2017), or occupancy functions (Mescheder et al., 2019), which use neural networks to represent 3D spatial functions. It would be desirable to fuse these more flexible shape representations with the discrete part structure of the ShapeVAE, ideally learning about part structure without requiring full supervision. One possible option is to associate local shape features with a latent part label, that could be observed or not observed for different instances. The semi-supervised lower bound described in Kingma et al. (2014) provides a principled objective, where inference over part segmentation is equivalent to latent variable inference.

In Chapter 4 we identified challenges to performing efficient inference in the presence of missing data for linear subspace models like factor analysis. We also

identified certain approximate inference methods that are more favourable. We cast inference as a constraint process, whereby each data variable provides an additional constraint on the latent space distribution. It would be desirable to incorporate latent constraints in a similar way to perform inference for nonlinear latent variable models. Vedantam et al. (2018) describe one approach, where each data variable contributes a component of a product of Gaussians. This naturally gives observed data the ability to 'veto' parts of latent space that are inconsistent with that observation. However the use of a product of Gaussians means that the model can only express linear constraints, which are insufficient in many cases. Normalizing flows are effective way to achieve expressive posterior distributions in latent variable models (Rezende and Mohamed, 2015; Kingma et al., 2016). In order to handle latent variable inference with missing data, each visible data variable could contribute a transformation in a normalizing flow, achieving a similar vetoing effect to the product of Gaussians, but with more expressive constraints.

The Multi-entity VAE presented in Chapter 5 was effective at learning to decompose scenes into multiple objects, however the scenes the model was tested on were quite simple: 2D, with well defined shapes, and a static black background. A number of works have made efforts to extend object-centric models to more complex scenes including 3D objects (Engelcke et al., 2020; Burgess et al., 2019). But, the domains in which these models are effective are still quite limited, with plain coloured geometric objects clearly highlighted in indoor environments. Future work would extend the model to work with more complex, and ideally natural scenes. This presents some challenges: an ability to localise the objects clearly against the background makes it more straightforward to obtain distinct object encodings, but in natural scenes the boundaries between objects are less clear. It may be the case that we require additional sources of information to localise objects, like temporal stability across scenes for instance. An alternative research direction is to explore extensions to other domains, such as speech. In this case the goal the informational attention mechanism could be adapted to operate on temporal signals rather than spatial, and it may be effective in capturing atomic units of speech, like phonemes or words.

As discussed in Chapter 6, it would be of interest to use inversion models to analyze a number of deep learning methods: dropout, batch normalization, residual

connections, and stochastic optimization. In doing so we could get a better sense of how these methods impact the representations learned by network, both in terms of mutual information with respect to the inputs, and through a visualisation of the network's invariances. Recently a family of alternative approaches to mutual information estimation have emerged, that are effective at estimating mutual information in some settings (Poole et al., 2019). Mutual information neural estimation (Belghazi et al., 2018, MINE) estimates mutual information with a discriminator that aims to distinguish between independent samples of two variables, and samples from their joint distribution. The extent to which this is possible determines the mutual information. This approach is promising, as discriminator based approaches are more lightweight than the generative approaches used by our model. However, there are fundamental limits to the amount of mutual information that can be detected through such approaches, which are determined by the number of samples observed (McAllester and Stratos, 2018).

Bibliography

T. W. Anderson and H. Rubin. Statistical inference in factor analysis. In *Proceedings of the third Berkeley symposium on mathematical statistics and probability*, volume 5, pages 111–150, 1956.

D. Arpit, Y. Zhou, H. Q. Ngo, and V. Govindaraju. Why regularized auto-encoders learn sparse representation? In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 136–144. JMLR.org, 2016.

M. Averkiou, V. G. Kim, Y. Zheng, and N. J. Mitra. ShapeSynth: Parameterizing model collections for coupled shape exploration and synthesis. *Computer Graphics Forum*, 33(2):125–134, 2014.

M. I. Belghazi, A. Baratin, S. Rajeswar, S. Ozair, Y. Bengio, R. D. Hjelm, and A. C. Courville. Mutual information neural estimation. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 530–539. PMLR, 2018.

Y. Bengio, A. C. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

C. M. Bishop. Mixture density networks. *Technical Report, NCRG/4288*, 1994.

C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

Y. Burda, R. B. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. In *ICLR*, 2016.

C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner. MONet: Unsupervised scene decomposition and representation. *CoRR*, abs/1901.11390, 2019.

L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *NeurIPS*, pages 2172–80, 2016.

X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel. Variational lossy autoencoder. In *ICLR*, 2017.

R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

C. B. Choy, M. Stark, S. Corbett-Davies, and S. Savarese. Enriching object detection with 2D-3D registration and continuous viewpoint estimation. In *CVPR*, pages 2512–2520. IEEE Computer Society, 2015.

M. Collier, A. Nazábal, and C. K. Williams. VAEs in the presence of missing data. *Workshop on the Art of Learning with Missing Values, ICML*, 2020.

T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models—their training and application. *Computer Vision and Image Understanding*, 61 (1):38–59, 1995.

T. M. Cover and J. A. Thomas. *Elements of Information Theory (2. ed.)*. Wiley, 2006.

C. Cremer, X. Li, and D. Duvenaud. Inference suboptimality in variational autoencoders. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1086–1094. PMLR, 2018.

A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A.

Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

L. N. Darlow and A. J. Storkey. What information does a ResNet compress? *CoRR*, abs/2003.06254, 2020.

E. Denton and R. Fergus. Stochastic video generation with a learned prior. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1182–1191. PMLR, 2018.

E. L. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep generative image models using a Laplacian pyramid of adversarial networks. In *NeurIPS*, pages 1486–1494, 2015.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186. Association for Computational Linguistics, 2019.

J. J. DiCarlo and D. D. Cox. Untangling invariant object recognition. *Trends in Cognitive Sciences*, 11(8):333–341, 2007.

N. Dilokthanakul, P. A. M. Mediano, M. Garnelo, M. C. H. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan. Deep unsupervised clustering with Gaussian mixture variational autoencoders. *CoRR*, abs/1611.02648, 2016.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. In *ICLR*, 2017.

C. Doersch. Tutorial on variational autoencoders. *CoRR*, abs/1606.05908, 2016.

A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. pages 4829–4837, 2016a.

A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In *NeurIPS*, pages 658–666, 2016b.

S. Dray and J. Josse. Principal components analysis with missing values: a comparative survey of methods. *Plant Ecology*, 216:657–667, 2015.

C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural spline flows. In *NeurIPS*, pages 7509–7520, 2019.

M. Engelcke, A. R. Kosiorek, O. P. Jones, and I. Posner. GENESIS: generative

scene inference and sampling with object-centric latent representations. *ICLR*, 2020.

S. M. A. Eslami, N. Heess, C. K. I. Williams, and J. M. Winn. The shape Boltzmann machine: A strong model of object shape. *International Journal of Computer Vision*, 107(2):155–176, 2014.

S. M. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, K. Kavukcuoglu, and G. E. Hinton. Attend, Infer, Repeat: Fast scene understanding with generative models. In *NeurIPS*, pages 3225–3233, 2016.

H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3D object reconstruction from a single image. In *CVPR*, pages 2463–2471. IEEE Computer Society, 2017.

N. Fish, M. Averkiou, O. van Kaick, O. Sorkine-Hornung, D. Cohen-Or, and N. J. Mitra. Meta-representation of shape families. *ACM Transactions on Graphics*, 33(4):34:1–34:11, 2014.

M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked autoencoder for distribution estimation. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 881–889. JMLR.org, 2015.

R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *ECCV (6)*, volume 9910 of *Lecture Notes in Computer Science*, pages 484–499. Springer, 2016.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, 2016.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014.

A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. DRAW: A recurrent neural network for image generation. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1462–1471. JMLR.org, 2015.

T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. A papier-mâché

approach to learning 3D surface generation. In *CVPR*, pages 216–224. IEEE Computer Society, 2018.

I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taïga, F. Visin, D. Vázquez, and A. C. Courville. PixelVAE: A latent variable model for natural images. In *ICLR*, 2017.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

T. Heimann and H. Meinzer. Statistical shape models for 3D medical image segmentation: A review. *Medical Image Analysis*, 13(4):543–563, 2009.

M. Hendrikx, S. Meijer, J. V. D. Velden, and A. Iosup. Procedural content generation for games: A survey. *TOMCCAP*, 9(1):1:1–1:22, 2013.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

I. Higgins, D. Amos, D. Pfau, S. Racanière, L. Matthey, D. J. Rezende, and A. Lerchner. Towards a definition of disentangled representations. *CoRR*, abs/1812.02230, 2018.

G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

H. Hoppe, T. DeRose, T. Duchamp, J. A. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH*, pages 71–78. ACM, 1992.

H. Huang, E. Kalogerakis, and B. M. Marlin. Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces. *Computer Graphics Forum*, 34(5):25–38, 2015.

X. Huang, Y. Li, O. Poursaeed, J. E. Hopcroft, and S. J. Belongie. Stacked generative adversarial networks. In *CVPR*, pages 1866–1875. IEEE Computer Society, 2017.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV (2)*, volume 9906 of *Lecture Notes in Computer Science*, pages 694–711. Springer, 2016.

J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, pages 1988–1997. IEEE Computer Society, 2017.

E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics*, 31(4): 55:1–55:11, 2012.

T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, pages 4401–4410. Computer Vision Foundation / IEEE, 2019.

M. M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing*, volume 256 of *ACM International Conference Proceeding Series*, pages 61–70. Eurographics Association, 2006.

V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. A. Funkhouser. Learning part-based templates from large collections of 3D shapes. *ACM Transactions on Graphics*, 32(4):70:1–70:12, 2013.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, pages 10236–10245, 2018.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *NeurIPS*, pages 3581–3589, 2014.

D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. In *NeurIPS*, 2016.

A. Kolchinsky and B. D. Tracey. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7):361, 2017.

A. Kolchinsky, B. D. Tracey, and D. H. Wolpert. Nonlinear information bottleneck. *CoRR*, abs/1705.02436, 2017.

A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical Review*, 69(6):066138, 2004.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. B. Tenenbaum. Deep convolutional inverse graphics network. In *NeurIPS*, pages 2539–2547, 2015.

A. Lamb, V. Dumoulin, and A. C. Courville. Discriminative regularization for generative models. *CoRR*, abs/1602.03220, 2016.

A. Lanitis, C. J. Taylor, and T. F. Cootes. Automatic interpretation and coding of face images using flexible models. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 19(7):743–756, 1997.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

S. Lee and R. M. Kil. Inverse mapping of continuous functions using local and global information. *IEEE Transactions on Neural Networks*, 5(3):409–423, 1994.

J. Liebelt and C. Schmid. Multi-view object class detection with a 3D geometric model. In *CVPR*, pages 1688–1695. IEEE Computer Society, 2010.

M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

A. Linden and J. Kindermann. Inversion of multilayer nets. *International Joint Conference on Neural Networks*, 1989.

R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. Wiley, 1987.

G. Loaiza-Ganem and J. P. Cunningham. The continuous Bernoulli: Fixing a pervasive error in variational autoencoders. In *NeurIPS*, pages 13266–13276, 2019.

B. Lu, H. Kita, and Y. Nishikawa. Inverting feedforward neural networks using linear and nonlinear programming. *IEEE Transactions on Neural Networks*, 10 (6):1271–1290, 1999.

C. Ma, S. Tschiatschek, K. Palla, J. M. Hernández-Lobato, S. Nowozin, and C. Zhang. EDDI: efficient dynamic discovery of high-value information with partial VAE. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 4234–4243. PMLR, 2019.

D. J. C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, pages 5188–5196. IEEE Computer Society, 2015.

K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Academic Press, London, 1979.

L. Matthey, I. Higgins, D. Hassabis, and A. Lerchner. dSprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

D. McAllester and K. Stratos. Formal limitations on the measurement of mutual information. *CoRR*, abs/1811.04251, 2018.

R. Memisevic and G. E. Hinton. Unsupervised learning of image transformations. In *CVPR*. IEEE Computer Society, 2007.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation*, 22(6): 1473–1492, 2010.

L. M. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, pages 4460–4470. Computer Vision Foundation / IEEE, 2019.

V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu. Recurrent models of visual attention. In *NeurIPS*, pages 2204–2212, 2014.

A. Mohamed, G. E. Dahl, and G. E. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions Audio, Speech & Language Processing*, 20(1): 14–22, 2012.

C. Nash and C. K. I. Williams. The shape variational autoencoder: A deep generative model of part-segmented 3D objects. *Computer Graphics Forum*, 36 (5):1–12, 2017.

C. Nash, S. A. Eslami, C. Burgess, I. Higgins, D. Zoran, T. Weber, and P. Battaglia. The multi-entity variational autoencoder. *NeurIPS Learning Disentangled Features workshop*, 2017.

C. Nash, N. Kushman, and C. K. Williams. Inverting supervised representations with autoregressive neural density models. In *AISTATS*, Proceedings of Machine Learning Research, 2019.

A. Nazábal, P. M. Olmos, Z. Ghahramani, and I. Valera. Handling incomplete heterogeneous data using VAEs. *CoRR*, abs/1807.03653, 2018.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. *Deep Learning and Unsupervised Feature Learning Workshop, NeurIPS*, 2011.

G. Papamakarios, I. Murray, and T. Pavlakou. Masked autoregressive flow for density estimation. In *NeurIPS*, pages 2338–2347, 2017.

G. Papamakarios, E. T. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *CoRR*, abs/1912.02762, 2019.

D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, pages 2536–2544. IEEE Computer Society, 2016.

N. Payet and S. Todorovic. From contours to 3D object detection and pose estimation. In *ICCV*, pages 983–990. IEEE Computer Society, 2011.

B. Poole, S. Ozair, A. van den Oord, A. Alemi, and G. Tucker. On variational bounds of mutual information. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 5171–5180. PMLR, 2019.

C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, pages 77–85. IEEE Computer Society, 2017.

T. Rainforth, A. R. Kosiorek, T. A. Le, C. J. Maddison, M. Igl, F. Wood, and Y. W.

Teh. Tighter variational bounds are not necessarily better. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 4274–4282. PMLR, 2018.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1530–1538. JMLR.org, 2015.

D. J. Rezende and F. Viola. Taming VAEs. *CoRR*, abs/1810.00597, 2018.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1278–1286. JMLR.org, 2014.

L. Romaszko, C. K. I. Williams, P. Moreno, and P. Kohli. Vision-as-Inverse-Graphics: Obtaining a rich 3D explanation of a scene from a single image. In *ICCV Workshops*, pages 940–948. IEEE Computer Society, 2017.

N. L. Roux, N. Heess, J. Shotton, and J. M. Winn. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011.

S. T. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11(2):305–345, 1999.

D. B. Rubin and D. T. Thayer. EM algorithms for ML factor analysis. *Psychometrika*, 47(1):69–76, 1982.

R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *AISTATS*, volume 5 of *JMLR Proceedings*, pages 448–455. JMLR.org, 2009.

R. Salakhutdinov, S. T. Roweis, and Z. Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *ICML*, pages 672–679. AAAI Press, 2003.

T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*, page 901, 2016.

T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017.

A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia,

and T. P. Lillicrap. A simple neural network module for relational reasoning. *NeurIPS*, 2017.

L. K. Saul, T. S. Jaakkola, and M. I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. In *ICLR*, 2018.

R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.

K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *NeurIPS*, pages 3483–3491, 2015.

C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. Ladder variational autoencoders. In *NeurIPS*, pages 3738–3746, 2016.

E. S. Spelke and K. D. Kinzler. Core knowledge. *Developmental science*, 10(1): 89–96, 2007.

H. Su, Q. Huang, N. J. Mitra, Y. Li, and L. J. Guibas. Estimating image depth using shape collections. *ACM Transactions on Graphics*, 33(4):37:1–37:11, 2014.

X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:421425:1–421425:19, 2009.

R. W. Sumner, J. Schmid, and M. Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics*, 26(3):80, 2007.

M. Sundermeyer, R. Schlüter, and H. Ney. LSTM neural networks for language modeling. In *INTERSPEECH*, pages 194–197. ISCA, 2012.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NeurIPS*, pages 3104–3112, 2014.

Y. Tang, R. Salakhutdinov, and G. E. Hinton. Deep mixtures of factor analysers. In *ICML*. icml.cc / Omnipress, 2012.

M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. In *ICCV*, pages 2107–2115. IEEE Computer Society, 2017.

J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12(6):1247–1283, 2000.

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. 2016.

M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61 (3):611–622, 1999.

R. Turner, M. Sahani, D. Barber, A. Cemgil, and S. Chiappa. Two problems with variational expectation maximisation for time-series models. *Bayesian Time Series Models*, pages 109–130, 01 2011.

B. Uria, I. Murray, and H. Larochelle. RNADE: the real-valued neural autoregressive density-estimator. In *NeurIPS*, pages 2175–2183, 2013.

B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 467–475. JMLR.org, 2014.

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *ISCA Speech Synthesis Workshop*, 2016a.

A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with PixelCNN decoders. In *NeurIPS*, pages 4790–4798, 2016b.

A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1747–1756. JMLR.org, 2016c.

A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, pages 6306–6315, 2017.

L. van der Maaten, E. Postma, and J. van den Herik. Dimensionality reduction: A comparative review. *JMLR*, 10(66-71):13, 2009.

M. J. van Kreveld, T. van Lankveld, and R. C. Veltkamp. On the shape of a set of points and lines in the plane. *Computer Graphics Forum*, 30(5):1553–1562, 2011.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

R. Vedantam, I. Fischer, J. Huang, and K. Murphy. Generative models of visually grounded imagination. In *ICLR*, 2018.

P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 1096–1103. ACM, 2008.

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *CVPR*, pages 3156–3164. IEEE Computer Society, 2015.

N. Watters, D. Zoran, T. Weber, P. W. Battaglia, R. Pascanu, and A. Tacchetti. Visual Interaction Networks: Learning a physics simulator from video. In *NeurIPS*, pages 4539–4547, 2017.

G. C. Wei and M. A. Tanner. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American statistical Association*, 85(411):699–704, 1990.

C. K. Williams, C. Nash, and A. Nazábal. Autoencoders and probabilistic inference with missing data: An exact solution for the factor analysis case. *CoRR*, abs/1801.03851, 2018.

C. K. I. Williams and F. V. Agakov. Products of Gaussians and probabilistic minor component analysis. *Neural Computation*, 14(5):1169–1182, 2002.

M. A. Woodbury and M. Woodbury. Inverting modified matrices. In *Memorandum Rept. 42*. Princeton University Statistical Research Group, 1950.

J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NeurIPS*, pages 82–90, 2016.

J. Wu, J. B. Tenenbaum, and P. Kohli. Neural scene de-rendering. In *CVPR*, pages 7035–7043. IEEE Computer Society, 2017.

Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920. IEEE Computer Society, 2015.

M. E. Yümer, P. Asente, R. Mech, and L. B. Kara. Procedural modeling using autoencoder networks. In *UIST*, pages 109–118. ACM, 2015.

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV (1)*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer, 2014.

Y. Zheng, D. Cohen-Or, and N. J. Mitra. Smart Variations: Functional substructures for part compatibility. *Computer Graphics Forum*, 32(2):195–204, 2013.

M. Z. Zia, M. Stark, B. Schiele, and K. Schindler. Detailed 3D representations for object recognition and modelling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(11):2608–2623, 2013.

S. Zuffi and M. J. Black. The stitched puppet: A graphical model of 3D human shape and pose. In *CVPR*, pages 3537–3546. IEEE Computer Society, 2015.